Table of Contents

Android Studio 使用艺术

本书不一定要按照从头开始读到尾，选择自己需要的部分阅读，类似于我们Java的API文档，但是API文档更加容易理解，会有很多的使用技巧，使用经验在里面。

每一个操作都会带实例演示，让读者看到我的操作步骤。

此书采用开源的形式编写，所以每个Android都可以参与，而且这个不像某些技术形书籍，需要功底才能表达出来，此书属于操作类型的，没有什么高深的理论，只需要把你使用的方法写出来就可以。

书中提到的快捷键全部都是 Android Studio 默认快捷键。

所有的源文件都在 github 上，希望大伙一起来做这个有意思的事情。

交流群：`104848238`

阅读地址：https:\/\/as.quanke.name\/

下载地址：https:\/\/www.gitbook.com\/book\/quanke\/android-studio\/details

协作地址：https:\/\/github.com\/quanke\/android-studio

我是全科，欢迎关注我的博客 http:\/\/quanke.name

> 此书不一定是按顺序编写

现在正在编辑的章节 ：

已经完成编辑的章节：

Android Studio 2.2 正式稳定版

https://www.oschina.net/news/77286/android-studio-2-2-stable

http:\/\/www.oschina.net\/news\/77431\/android-studio-2-2-new-features

前言

很多做Android开发的同学都关注技术，但我感觉对于一个开发者来说，工具的使用非常重要，可以提高编码速度，可以让更多的时间花在逻辑上，或者有更多的时间成长，而不是浪费在敲代码上，对于一个管理者来说更加重要，工具的熟练成功直接影响开发效率，所以我有编辑此书的想法。

第一章 Android Studio 入门

本章主要是介绍在Windows，Linux ，Mac 三个平台下载安装Android Studio。

下载

官方下载

http://developer.android.com/sdk/index.html

这里提供三个平台下载，请选择需要的下载

选择其他平台

平台	Android Studio 软件包	大小	SHA-1 校验和
Windows	android-studio-bundle-143.3101438-windows.exe 包含 Android SDK（**推荐**）	1262 MB (1324294792 bytes)	10d319c772b80f3cb0cde952451af8429ea1b68b
	android-studio-ide-143.3101438-windows.exe 无 Android SDK	259 MB (271696304 bytes)	43f84de7e61f37880a126c3d567b7fa6cb90c90e
	android-studio-ide-143.3101438-windows.zip 无 Android SDK，无安装程序	275 MB (289169874 bytes)	8ad212c55c7f4dc7ab490e4b7e77ec48001ae224
Mac OS X	android-studio-ide-143.3101438-mac.dmg	273 MB (287207166 bytes)	06166759b0e1e1ee91a147dcf5227d897a184277
Linux	android-studio-ide-143.3101438-linux.zip	273 MB (286664165 bytes)	8729e6f2f1fa58f04df9f8d1caac2f5be9dfc549

请参阅 Android Studio 发行说明。

国内下载

虽然官方的下载是最权威，最新的，但是考虑国内的情况，有同仁提供国内下载。

http://www.androiddevtools.cn/

此在下不一定是最新的，并且所有的，个人建议还是走官方渠道下载。

Windows 环境配置

配置JDK

下载JDK8

下载地址：http:\/\/www.oracle.com\/technetwork\/java\/javase\/downloads\/jdk8-downloads-2133151.html

Java SE Development Kit 8u101

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.77 MB	jdk-8u101-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.72 MB	jdk-8u101-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.28 MB	jdk-8u101-linux-i586.rpm
Linux x86	174.96 MB	jdk-8u101-linux-i586.tar.gz
Linux x64	158.27 MB	jdk-8u101-linux-x64.rpm
Linux x64	172.95 MB	jdk-8u101-linux-x64.tar.gz
Mac OS X	227.36 MB	jdk-8u101-macosx-x64.dmg
Solaris SPARC 64-bit	139.66 MB	jdk-8u101-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.96 MB	jdk-8u101-solaris-sparcv9.tar.gz
Solaris x64	140.33 MB	jdk-8u101-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u101-solaris-x64.tar.gz
Windows x86	188.32 MB	jdk-8u101-windows-i586.exe
Windows x64	193.68 MB	jdk-8u101-windows-x64.exe

选择自己合适的平台，下载

安装JDK

下载完成后双击安装,并按照提示一步一步直到完成安装。

> 建议默认安装

安装时请记住JDK的安装位置,后面我们配置环境变量的时候要用到。

配置环境变量

右击计算机 —> 属性 —> 高级系统设置 —> 环境变量

新建系统变量JAVA_HOME —> 变量值为刚才安装的JDK的路径

在系统变量 `path` 中添加 `%JAVA_HOME%\bin`

新建系统变量 `CLASS_PATH` —> 添加变量值 `%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;`

注意：如果 `CLASS_PATH` 或者 `path` 中有其他配置，请在后面增加 `;` 再增加我们需要的内容，尤其是CLASS_PATH 需要注意

到此为止环境变量配置完毕,下面来验证下是否配置成功.

终端输入 `javac` ,如果显示帮助信息就证明配置正确了.

```
C:\Users\yurong>javac

用法: javac <options> <source files>

其中, 可能的选项包括:

 -g 生成所有调试信息

 -g:none 不生成任何调试信息

 -g:{lines,vars,source} 只生成某些调试信息

 -nowarn 不生成任何警告

 -verbose 输出有关编译器正在执行的操作的消息

 -deprecation 输出使用已过时的 API 的源位置

 -classpath <路径> 指定查找用户类文件和注释处理程序的位置

 -cp <路径> 指定查找用户类文件和注释处理程序的位置

 -sourcepath <路径> 指定查找输入源文件的位置

 -bootclasspath <路径> 覆盖引导类文件的位置

 -extdirs <目录> 覆盖所安装扩展的位置

 -endorseddirs <目录> 覆盖签名的标准路径的位置

 -proc:{none,only} 控制是否执行注释处理和/或编译。

 -processor <class1>[,<class2>,<class3>...] 要运行的注释处理程序的名称; 绕过默

认的搜索进程

 -processorpath <路径> 指定查找注释处理程序的位置

 -parameters 生成元数据以用于方法参数的反射

 -d <目录> 指定放置生成的类文件的位置

 -s <目录> 指定放置生成的源文件的位置
```

```
-h <目录> 指定放置生成的本机标头文件的位置

-implicit:{none,class} 指定是否为隐式引用文件生成类文件

-encoding <编码> 指定源文件使用的字符编码

-source <发行版> 提供与指定发行版的源兼容性

-target <发行版> 生成特定 VM 版本的类文件

-profile <配置文件> 请确保使用的 API 在指定的配置文件中可用

-version 版本信息

-help 输出标准选项的提要

-A关键字[=值] 传递给注释处理程序的选项

-X 输出非标准选项的提要

-J<标记> 直接将 <标记> 传递给运行时系统

-Werror 出现警告时终止编译

@<文件名> 从文件读取选项和文件名
```

配置Android Studio

下载Android Studio

你可以到这里http:\/\/developer.android.com\/sdk\/index.html下载可执行文件进行安装.

也可以到这里下载最近的beta版本http:\/\/tools.android.com\/recent试用,beta版本不用安装,可以多个版本一起使用。

启动Android Studio

下载完成后 —> 解压 —> 进入到android\/bin目录 —> 双击studio64(如果操作系统是32位 双击 studio)启动Android Studio。

如果你想导入原来的设置,选中上面的复选框,如果不想就选上下面的复选框.

接下来就一步一步按照提示往下安装.

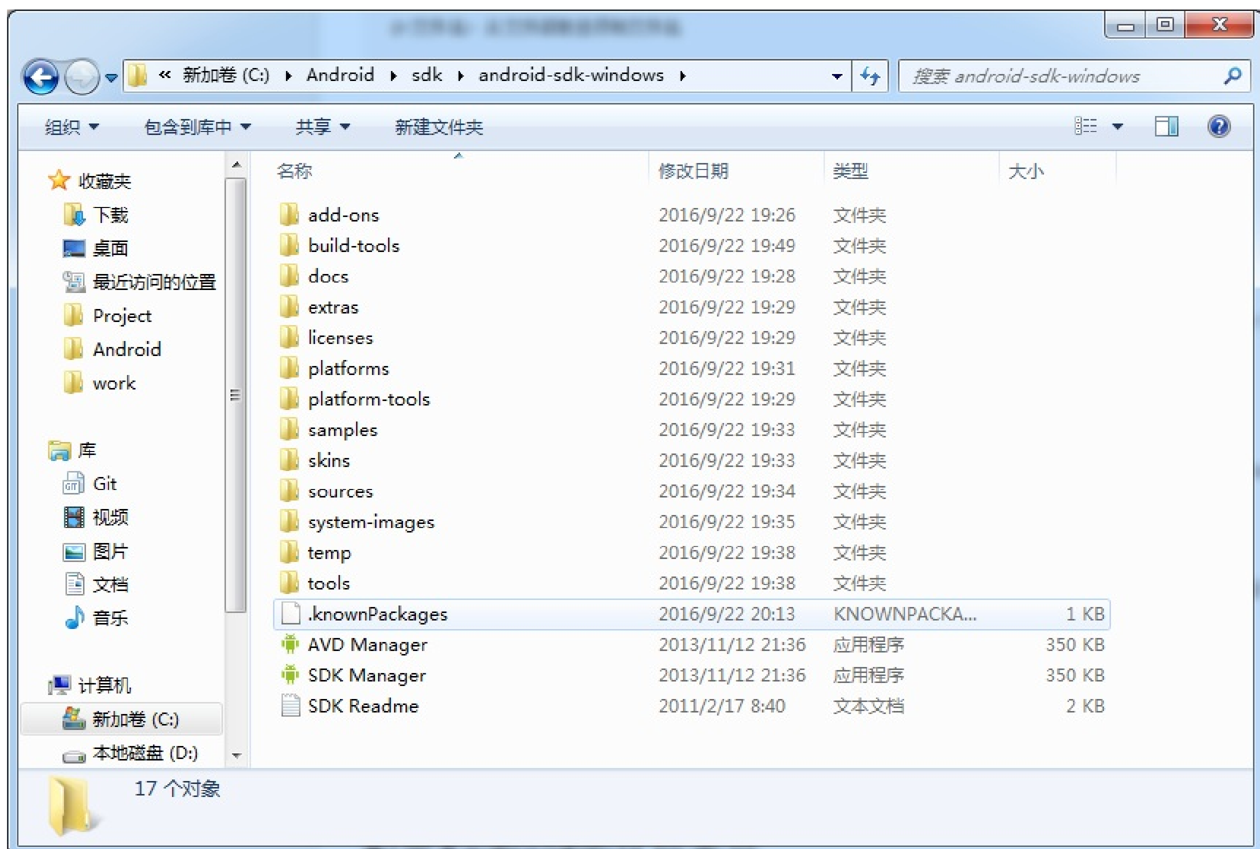
安装过程中会安装SDK和一些组件,请注意一下SDK的安装位置,如果你要在终端直接使用android命令还需要配置ANDROID的环境变量.

下载完成后就可以正常启动Android Studio啦.

配置Android的环境变量

如果我们想直接使用android提供的一些工具,就需要配置环境变量.

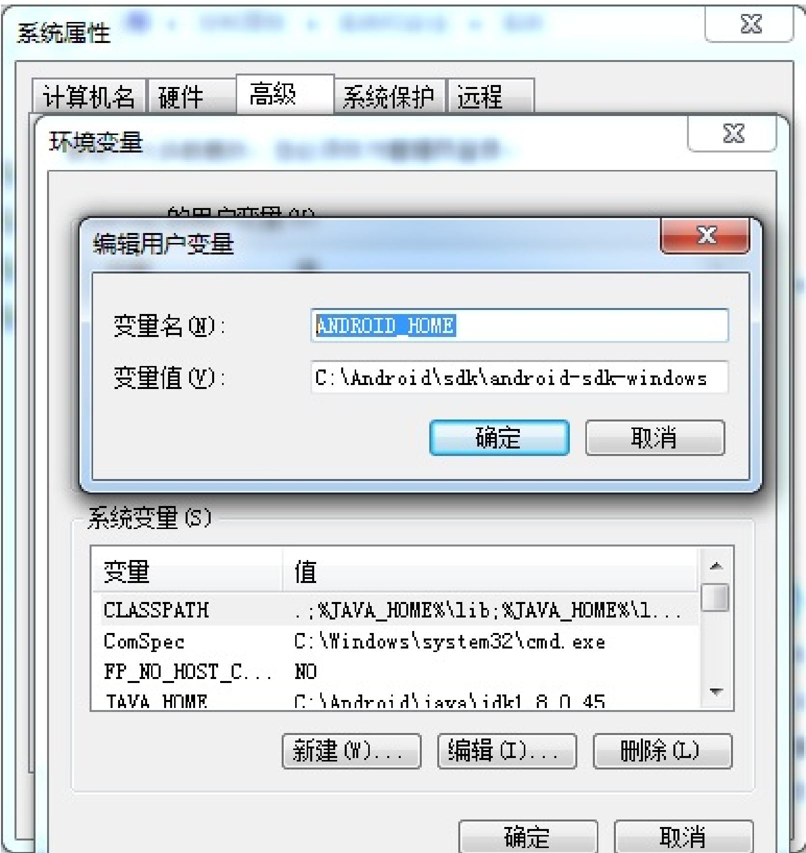
打开我们下载好的sdk目录,看一下里面的目录结构,有很多工具可供我们使用。



接下来我们配置下环境变量.

右击计算机 —> 属性 —> 高级系统设置 —> 环境变量 —> 新建系统变量ANDROID_HOME —> 变量值为刚才安装的SDK的路径

接下来在path中添加Android sdk和Android工具的路径



完成后验证一下是否配置的正确。

```
C:\Users\yurong>adb
Android Debug Bridge version 1.0.32
Revision 09a0d98bebce-android

 -a                            - directs adb to listen on all interfaces for a c
onnection
 -d                            - directs command to the only connected USB devic
e
                                 returns an error if more than one USB device is
 present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is r
unning.
 -s <specific device>          - directs command to the device or emulator with
the given
                                 serial number or qualifier. Overrides ANDROID_S
ERIAL
                                 environment variable.
 -p <product name or path>     - simple product name like 'sooner', or
                                 a relative/absolute path to a product
                                 out directory like 'out/target/product/sooner'.

                                 If -p is not specified, the ANDROID_PRODUCT_OUT

                                 environment variable is used, which must
                                 be an absolute path.
 -H                            - Name of adb server host (default: localhost)
 -P                            - Port of adb server (default: 5037)
 devices [-l]                  - list all connected devices
                                 ('-l' will also list device qualifiers)
 connect <host>[:<port>]       - connect to a device via TCP/IP
                                 Port 5555 is used by default if no port number
is specified.
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device
```

只要不报adb不是内部命令就是配置成功啦。

> 如果还有任何问题请留言

Mac OS 环境配置

配置JDK

第**1**步: 下载**JDK**

下载地址: JDK8: http:\/\/www.oracle.com\/technetwork\/java\/javase\/downloads\/jdk8-downloads-2133151.html

> Android Studio2.2以后要求必须使用JDK8。

选择Mac OS X x64下载:

Java SE Development Kit 8u101

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.77 MB	jdk-8u101-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.72 MB	jdk-8u101-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.28 MB	jdk-8u101-linux-i586.rpm
Linux x86	174.96 MB	jdk-8u101-linux-i586.tar.gz
Linux x64	158.27 MB	jdk-8u101-linux-x64.rpm
Linux x64	172.95 MB	jdk-8u101-linux-x64.tar.gz
Mac OS X	227.36 MB	jdk-8u101-macosx-x64.dmg
Solaris SPARC 64-bit	139.66 MB	jdk-8u101-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.96 MB	jdk-8u101-solaris-sparcv9.tar.gz
Solaris x64	140.33 MB	jdk-8u101-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u101-solaris-x64.tar.gz
Windows x86	188.32 MB	jdk-8u101-windows-i586.exe
Windows x64	193.68 MB	jdk-8u101-windows-x64.exe

第**2**步：安装**JDK8**

下载完成后，双击安装包，然后按照提示进行安装。

安装完成后的路径：

\/Library\/Java\/JavaVirtualMachines\/

如果你安装了多个JDK版本,这里会显示多个。

第**3**步：配置环境变量

先查看原来的java版本

```
java -version
```

再vim .bash_profile 去配置JAVA_HOME

```
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home

PATH=$JAVA_HOME/bin:$PATH

CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar



export JAVA_HOME

export CLASSPATH

export PATH
```

保存后去执行下面的命令，更新成功

```
source .bash_profile
```

```
java -version
```

配置Android Studio

1.下载**Android SDK**

到http:\/\/developer.android.com\/sdk\/index.html下载独立的Android SDK安装包.

我下载的是android-sdk_r24.4.1-macosx.zip.

2.解压安装包

下载完成后解压,放到你指定的目录

3.配置环境变量

vim ~\/.bash_profile把下面这些环境变量加进去

```
export ANDROID_HOME=SDK路径

export PATH=$ANDROID_HOME/platform-tools:$ANDROID_HOME/tools:$PATH
```

4.测试一下

命令行输入：

```
adb
```

Linux 环境配置

安装JDK

下载JDK8

下载地址:http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

先增加执行权限：sudo chmod 777 jdk-8u101-linux-x64.bin

新建一个文件夹：sudo mkdir /opt/java "这个是要注意的，因为**intel lij** 对路径的识别只支持三个路径，所有，要把**JDK**安装在这三个之一： `/usr/java` **or** `/opt/java` **or** `/usr/lib/jvm` 我就在这出了一个问题

移动到要安装的目录：sudo mv jdk-8u101-linux-x64.bin /opt/java

sudo /opt/java/jdk-8u101-linux-x64.bin

这个安装包的任务完成了，扫扫：sudo rm -f /opt/java/jdk-8u101-linux-x64.bin

配置JDK

```
sudo gedit ~/.bashrc 加上：

#set java environment

export JAVA\_HOME=\/opt\/java\/jdk1.8.0\_45

export JRE\_HOME=$JAVA\_HOME\/jre

export CLASSPATH=$CLASSPATH:$JAVA\_HOME\/lib:$JRE\_HOME\/lib

export PATH=$PATH:$JAVA\_HOME\/bin:$JRE\_HOME\/bin:$JAVA\_HOME\/lib:$JAVA\_HOME
```

`sudo gedit /etc/environment` 加上：

JAVA_HOME=/opt/java/jdk1.8.0_45 ”这个是我遇到第二个问题，不知道什么原因，最开始我是在/etc/profile配置额上边～/.bashrc加的那段，就是不行，找不到JAVA_HOME这个项，就直接放到这个文件了一份，ANDROID STUDIO就可以启动了。 之后又吧/etc\/profile的配置项挪到了～/.bashrc了， 我还很小白，现在还不知道是什么原因导致不能读到/etc/profile的配置项，猜测可能是没更新环境变量？有知道的帮忙回一下。

配置Android Studio

```
sudo mv android-studio-bundle-130.677228-linux.tgz /opt/ ″" //我直接用的文档管理器解压再挪过去的

sudo tar zxvf android-studio-bundle-130.677228-linux.tgz
```

可以将目录改成你喜欢的名字： `sudo mv /opt/and... /opt/android-studio`

```
cd /opt/android-studio/bin

ls

sudo studio.sh
```

版本更新

启动时检测版本更新

在欢迎界面 —> 点击底部的Check for updates now的Check.

手动检测版本更新

菜单栏:

Mac: Android Studio --> Check for Updates ...

Windows/Linux: Help —> Check for Updates ...然后Android Studio就会开始检测是否有更新。

如果已经是最新版本,就会弹出提示。

如果不是最新版本,就会弹出更新提示,可能是Android Studio版本更新,也有可能是插件或SDK更新。

配置更新通道

在偏好设置中配置更新通道

操作步骤: Preferences —> Appearance & Behavior —> System Settings —> Updates

在Updates界面上显示了当前版本的信息和上次检查更新的时间，我们可以在这里配置Android Studio和Android SDK的更新通道, 然后立即检查更新.

在检测结果对话框中配置更新通道

当我们执行检测更新以后,弹出的检测结果对话框中也是可以配置更新通道的。 操作步骤: 菜单栏 --> Android Studio --> Check for Updates ... 在弹出的对话框中选择Update进入更新通道配置界面,

单独下载最近更新的版本

下载地址: http://tools.android.com/download/studio

四种版本的区别

Android Studio提供了四种更新通道, 在检测更新时分别会检测不同的版本.

Stable Channel: 稳定版本

Beta Channel: 测试版本 Dev Channel: 开发版本

Canary Channel: 金丝雀版本

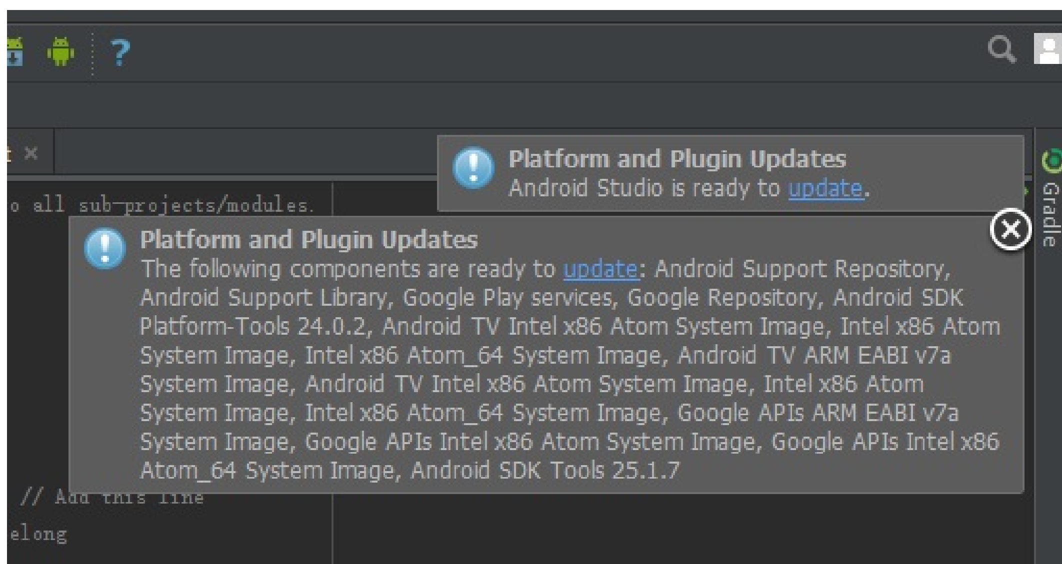
这四个版本更新频率从Canary往上逐渐递减,Canary大概1或者2周会更新一次，Beta则为相对稳定的发布版，而Stable则是正式版.

另外这四个版本的稳定性则跟其更新频率相反, 稳定性从Canary往上逐渐递增。

环境问题经常会经常困扰着我们,所以如果不是特别需要，我们当然是使用稳定版本了，如果你想尝鲜, 当然是使用Canary Channel.

Canary Channel最近的更新信息: http://tools.android.com/download/studio/canary/latest

提示平台和插件更新



Android Studio 项目和模块

Android Studio中的两个重要的概念: 项目(Project)和模块(Module) .

模块**(Module):**

Module是一个可以单独的运行和调试的application或公共库,它相当于Eclipse当中的Project.

我们通常把一些公共的代码放到module当中,当你的APP需要添加一些依赖的库或者需要依赖一些公共的项目时可以导入module.

每个module中都有自己的build.gradle文件用来配置模块的构建任务.

在build.gradle文件中我们可以配置SDK版本、构建工具版本、应用程序版本、打包参数、模块的依赖等等.

项目**(Project):**

Project可以理解为一个完整的APP项目,它由application module和一些依赖的module组成, 相当于Eclipse当中的workspace.

一个Project中有多个module。

Project中也有一个build.gradle文件用来指定构建的项目和任务,当你导入或新建一个Module时,build.gradle会自动更新.

项目 **VS** 模块

Module相当于Eclipse当中的Project.Project相当于Eclipse当中的workspace.一个Project中可以包含多个Module.Module中的build.grade用于配置模块的构建任务.Project中的build.grade用于定构建的项目和任务.

第二章 新建

Android Studio提供了非常方便的功能来新建和导入项目、模块、文件以及各种资源，它集成了非常丰富的模板来帮助我们快速创建不同类型的文件。熟悉这些模板和工具的使用，会大大提高我们的开发效率。

创建项目

首先，先指出Android Studio中的两个概念。 **Project** 和 **Module** 。在Android Studio中， **Project** 的真实含义是工作空间， **Module** 为一个具体的项目。

在 **Eclipse** 中，我们可以同时对多个 **Eclipse** 的 **Project** 进行同时编辑，这些 **Project** 在同一个 **workspace** 之中。在Android Studio中，我们可以同时对多个Android Studio的 **Module** 进行同时编辑，这些 **Module** 在同一个**Project** 之中。

Eclipse 的 **Project** 等同于Android Studio的 **Module** 。

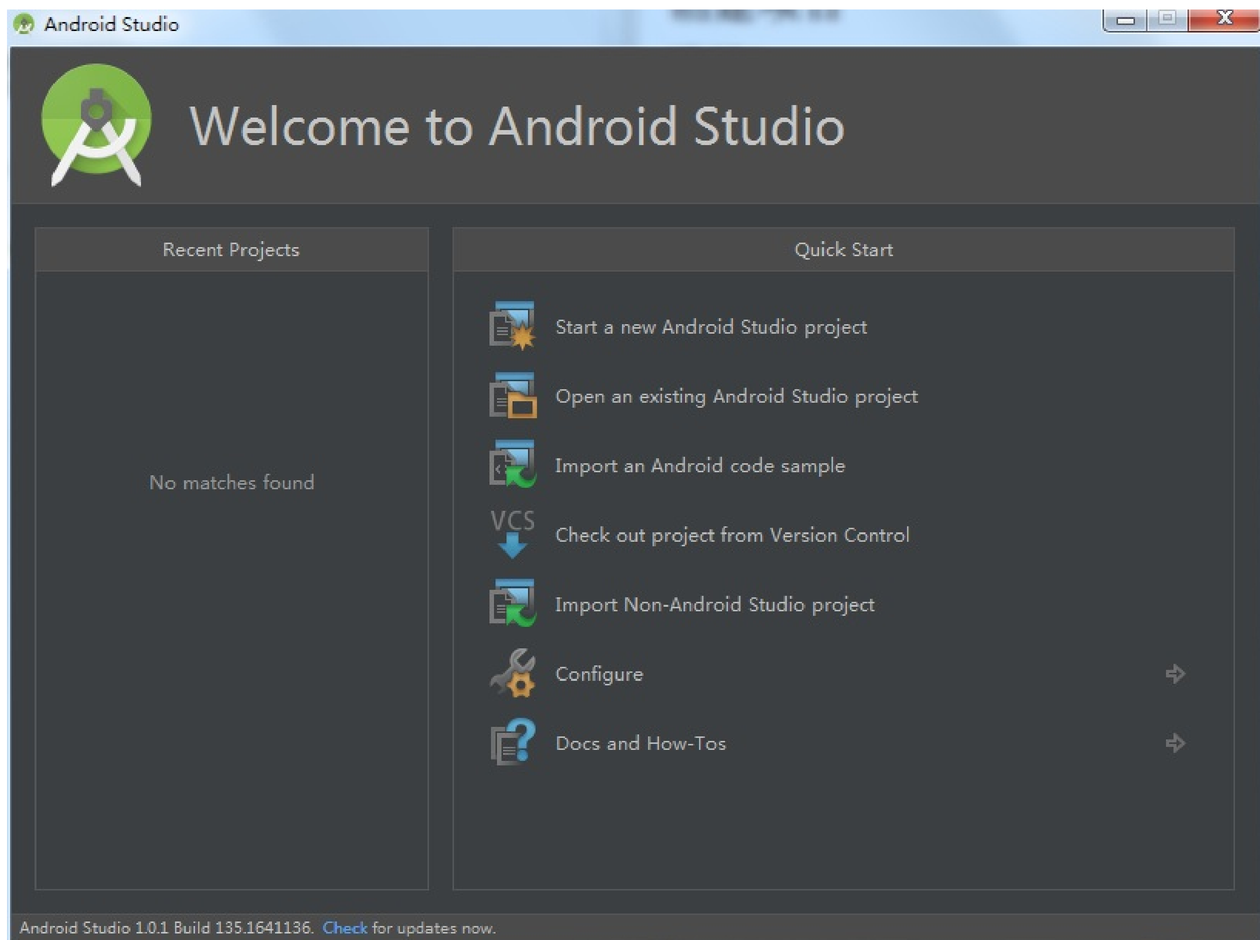
Eclipse 的 **workspace** 等同于Android Studio的 **Project** 。

本文中所说到的项目指的是Android Studio的 **Module** 。Android Studio创建一个项目，首先要先创建 **Project** 。但是你创建项目的同时， **Project** 自动创建了，因此很多人容易混淆这两种概念。

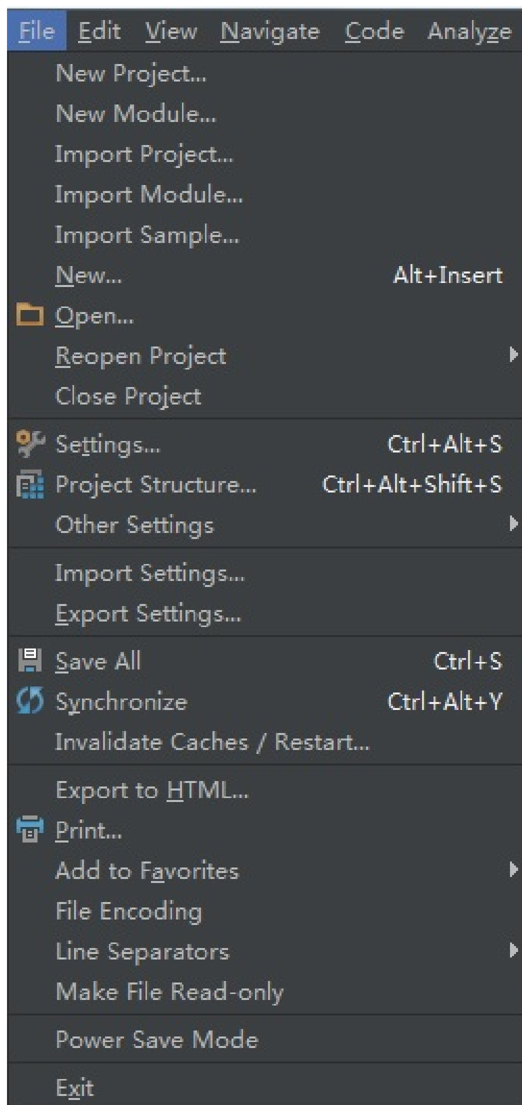
Android Studio创建项目的过程，其实就是 **Eclipse** 创建项目过程的细分化。 **Eclipse** 许多在一个页面设置的内容，Android Studio拆分成了多个页面，因此，创建项目的过程其实并不复杂。

Android Studio有两种创建项目的方法。

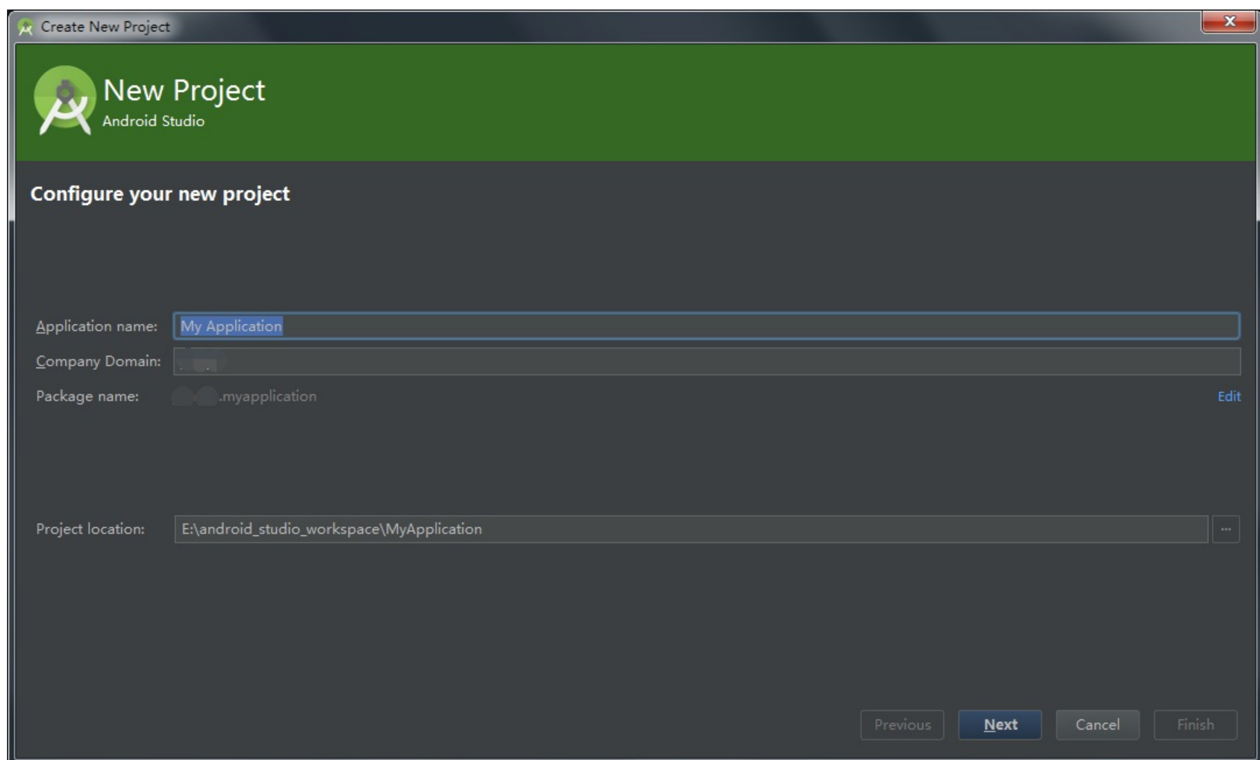
第一个是在Android Studio起始页选择 **Start a new Android Studio project** 。



第二个是在Android Studio主页，选择 **File --> New Project** 。



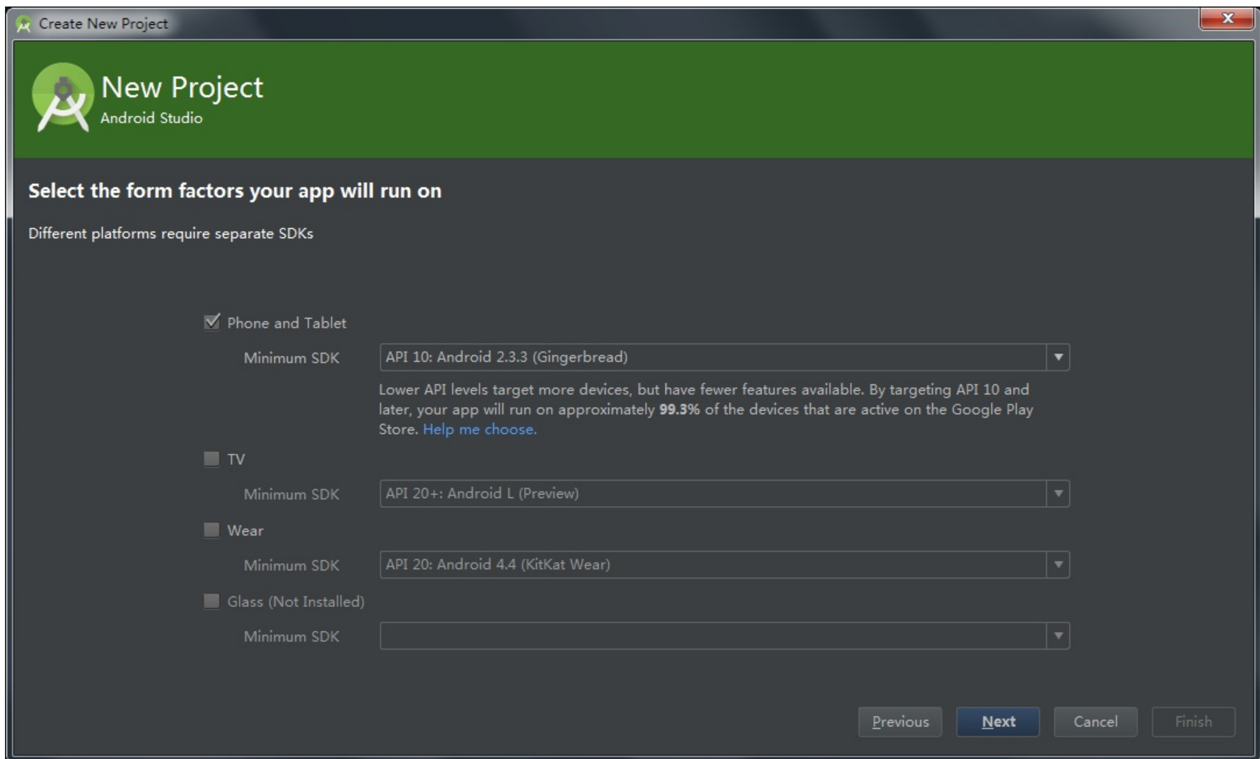
接下来，我们会看到这个页面。



- **Application name** ：应用程序的名称。它是app在设备上显示的应用程序名称，也是在Android Studio**Project** 的名称。
- **Company Domain** ：公司域名。影响下面的 **Package name** 。默认为电脑主机名称，当然你也可以单独设置 **Package name** 。
- **Package name** ：应用程序包名。每一个app都有一个独立的包名，如果两个app的包名相同，Android会认为他们是同一个app。因此，需要尽量保证，不同的app拥有不同的包名。
- **Project localtion** ： **Project** 存放的本地目录。

以上内容设置完毕，点击 **Next** 。

接下来，我们会看到这个页面。



在这里，你可以你的 **Project** 中 **Module** 的类型以及支持的最低版本。

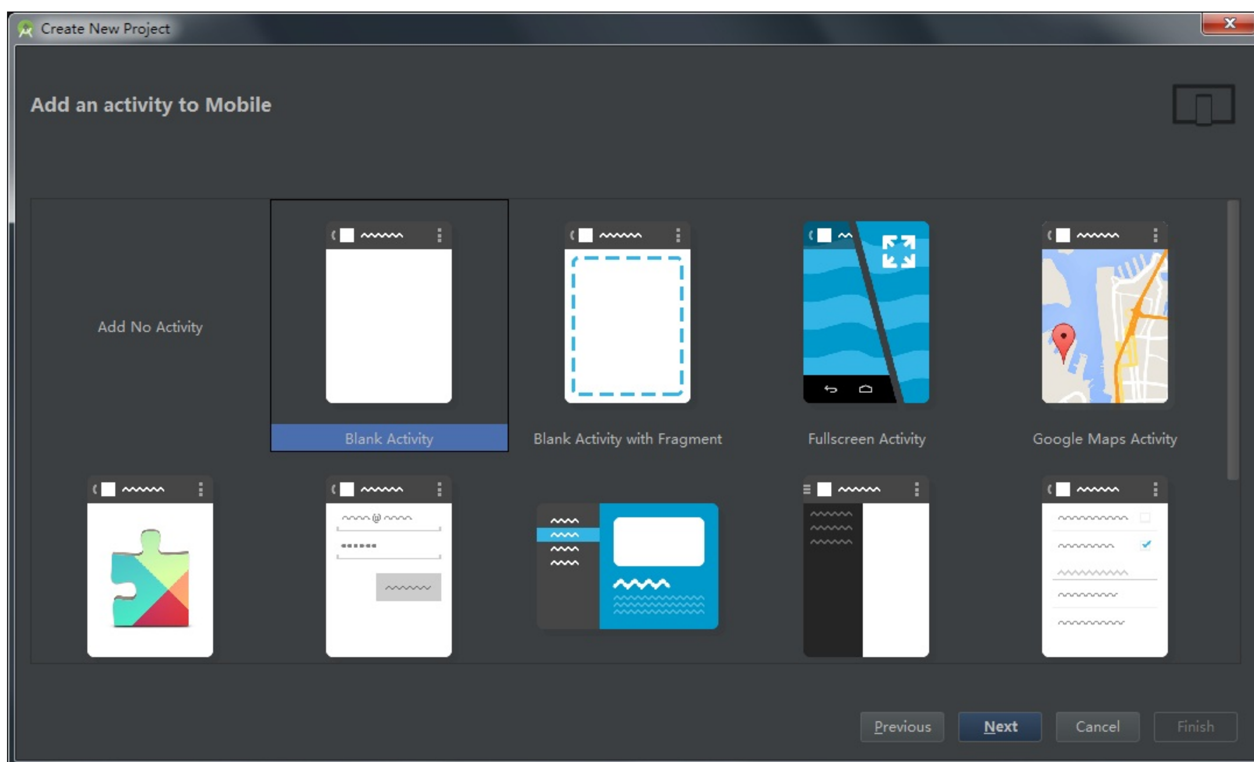
- **Phone and Tablet** ：表示 **Module** 是一个手机和平板项目。
- **TV** ：表示 **Module** 是一个Android TV项目。
- **Wear** ：表示 **Module** 是一个可穿戴设备（例如手表）项目。
- **Glass** ：表示 **Module** 是一个 **Google Glass** 项目（不知道 **Google Glass** 是什么请自行搜索）。

你可以同时选择多个类型，区别就是项目会根据你选择的类型创建一个或多个 **Module** 。

Minimum SDK 表示的是 **Module** 支持的Android最低版本。根据不同的用户可以选择不同的版本。你可以点击**Help me choose** 来查看当前Android版本分布情况。现在这个时代，如果你的项目支持到 **2.2** 版本几乎是支持了所有的Android设备。

以上内容设置完毕，点击 **Next** 。

接下来，我们会看到这个页面（由于我的 **Module** 类型只选择了 **Phone and Tablet** ，所以会有这个页面。）。

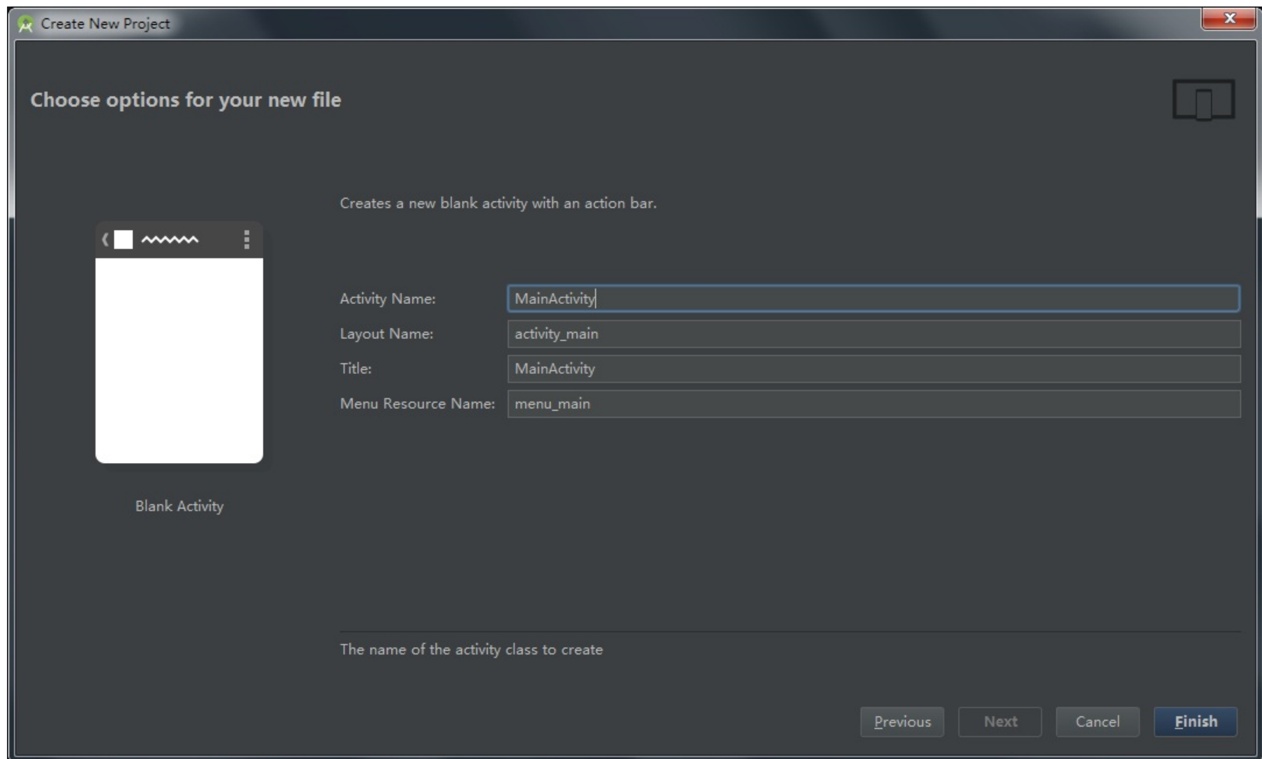


这个页面是让我们选择是否创建 **Activity**以及创建 **Activity** 的类型。你可以选择不创建 **Activity** （ **Add No Activity** ）。

> 如果你选择自动创建 **Activity**，Android Studio会自动帮你生成一些代码。根据 **Activity** 类型的不同，生成的代码也是不同的。有时，你能从这些自动生成的代码中，学到很多东西，比如 **Fullscreen Activity** 。

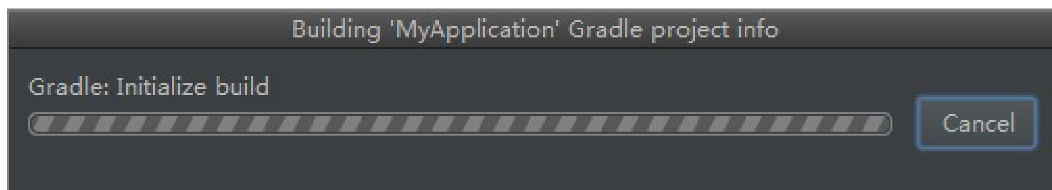
选择完毕，点击 **Next** 。

接下来，我们会看到这个页面（上一步中，我选择了 **Blank Activity** 。）。

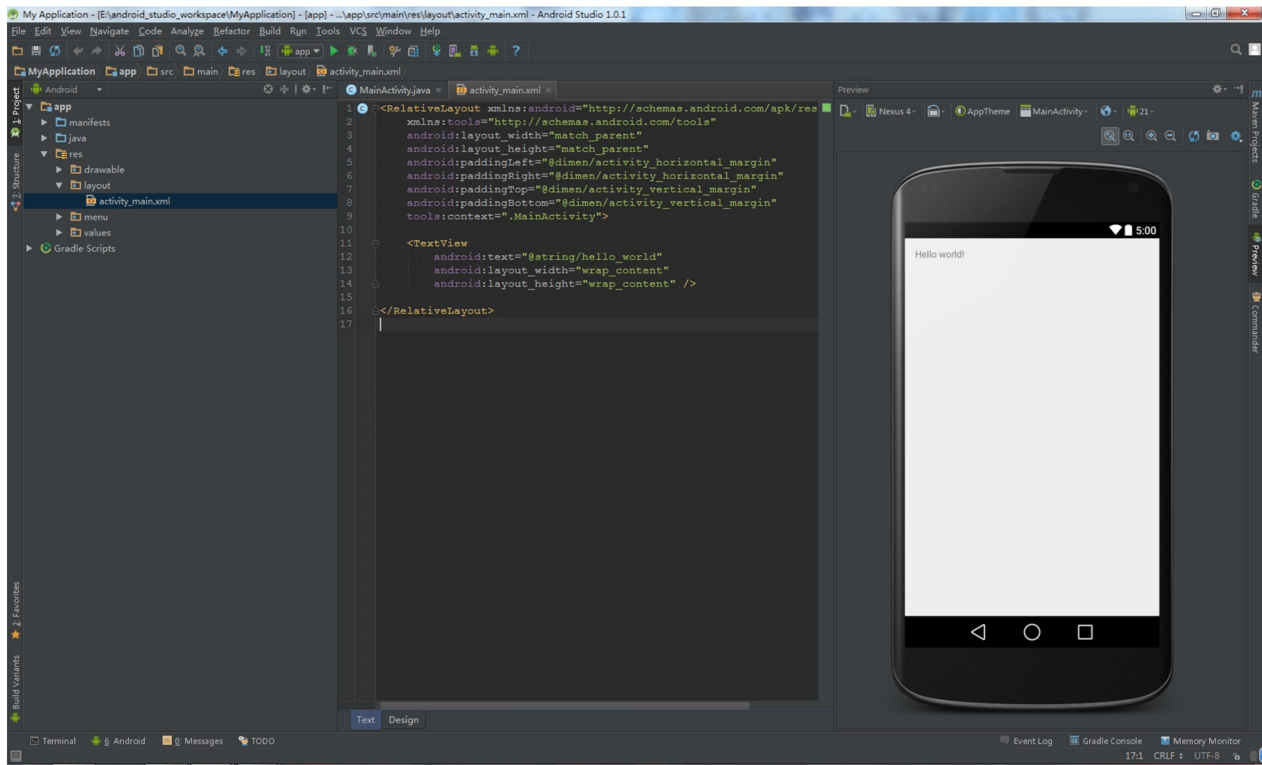


- **Activity Name** ：自动创建的 **Activity** 的类名。
- **Layout Name** ：自动创建的 **Activity** 的布局文件名称。
- **Title** :自动创建的 **Activity** 的名称。
- **Menu Resource Name** ：自动创建的 **Activity** 的 **Menu** 文件名称。

以上内容设置完毕，点击 **Next** 。你将会看到这个进度条。这个表示，Android Studio正在创建和编译你的项目。



项目创建编译完毕，会进入Android Studio的主页，你将看到下面的界面。 到这里项目已经创建完毕。

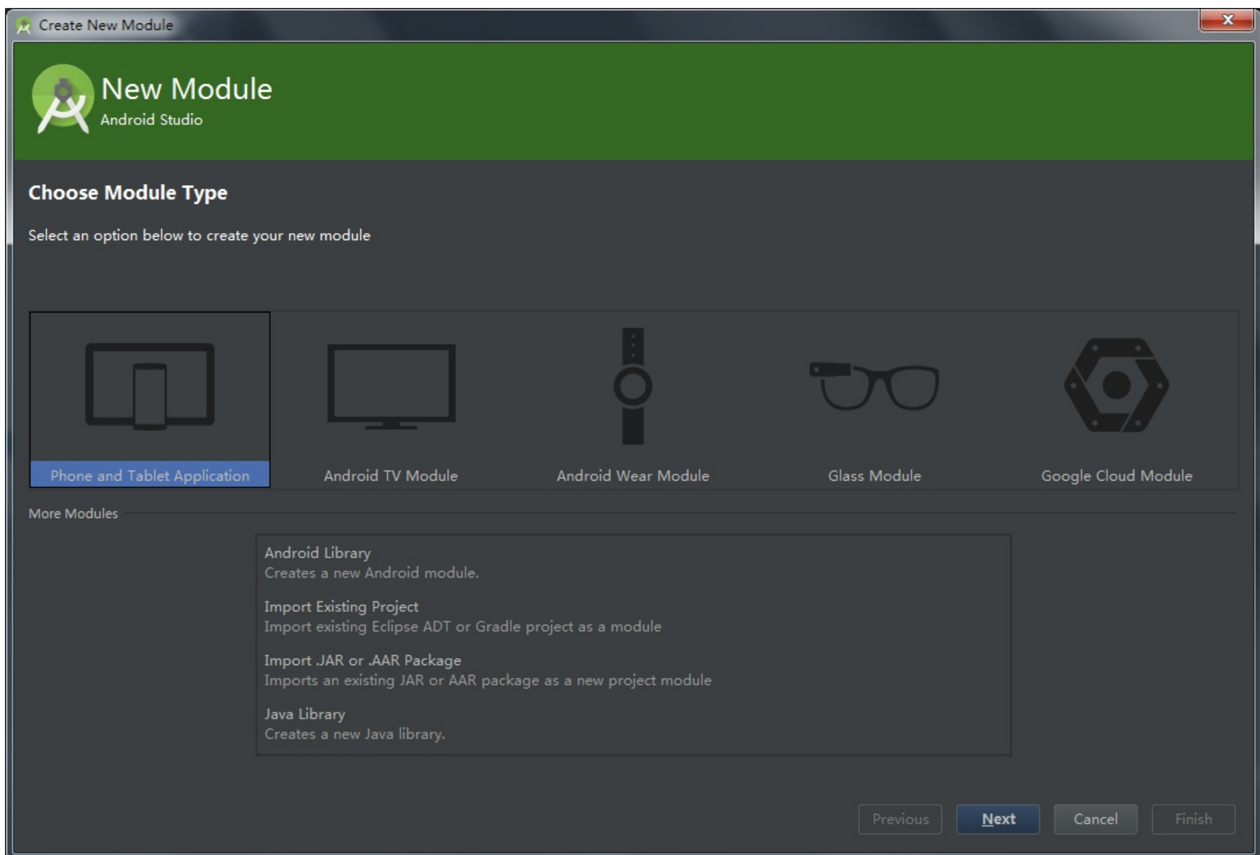

My Application - [E:\android_studio_workspace\MyApplication] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 1.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication app src main res layout activity_main.xml
MainActivity.java
activity_main.xml
app
manifests
java
res
drawable
layout
activity_main.xml
menu
values
Gradle Scripts
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
<TextView
android:text="@string/hello_world"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
</RelativeLayout>
Preview
Nexus 4
AppTheme
MainActivity
21
5:00
Hello world!
Text
Design
Terminal
6: Android
0: Messages
TODO
Event Log
Gradle Console
Memory Monitor

创建模块

如果你已经有一个项目，你想在这个项目中创建一个 **Module** ，你可以选择 **File --> New Module** 。

这样，会在当前的目录下创建一个 **Module** ，而不是在一个新的窗口中，单独创建 **Project** 和 **Module** 。

之后，你就会看到下面的界面。



同样的道理，你可以设置项目类型，但是，由于你现在是创建一个 **Module** ，所以只能选择一种类型。

下方的4个选项代表着不同的意义：

- **Android Library** ：创建一个 **Android Library** 的 **Module** 。
- **Import Existing Project** ：导入其他项目为一个 **Module** ，导入的项目可以是 **Eclipse ADT** 项目，也可以是**Gradle** 项目（ **Gradle** 只是项目构建工具， **Eclipse** 中也是可以使用的）。
- **Import .JAR or .AAR Package** ：导入 **JAR** 或者 **AAR** 文件为一个 **Module** 。
- **Java Library** ：创建一个 **Java** 的 **Module** ，主要用于编写 **Java** 的工具包。

选择完毕，点击 **Next** 。其余步骤，和创建项目几乎一样，就不在赘述了。

创建库

创建应用程序模块

操作步骤:

第1步:菜单栏: File —> New —> New Module

然后弹出创建模块界面

Android Studio中支持这么多种模块类型,我们要新建的模块类型为应用程序,因此要选择【Phone & Table Module】.

第2步: 选择[Phone & Table Module] —> Next —> 配置新的模块

第3步: Next —> 选择一个Activity模板

我们以Login Activity模板作为演示。

第4步:Next —> 配置Activity

这里有对即将创建的Activity模板的说明:

创建一个新的登录Activity, 允许用户选择使用Google + 登录,或输入email地址和密码登录,或注册你的APP.

我们这里使用默认配置.

第5步: Finish —> 新建应用程序(Phone & Table Module)模块成功

创建**Android**公共库模块

java的公共库是直接将公用代码打成jar包,android的公共库类似,目的都是为了代码的重用.使用android公共库还可以使项目模块化,以便协同开发和更好的扩展.

操作步骤**:**

菜单栏: File —>New —> New Module

然后弹出新建Module对话框,选择Android Library

Next —> 输入Library name和Module name

Finish —> 结果

为应用程序添加模块依赖:

操作步骤:

打开Project Structure —> 选择你要操作的应用程序 —> 添加已存在项目中的模块作为依赖

将Android Library打包成.aar

打开Gradle工具窗口,找到Android Library模块. 在 build任务中双击assemble.

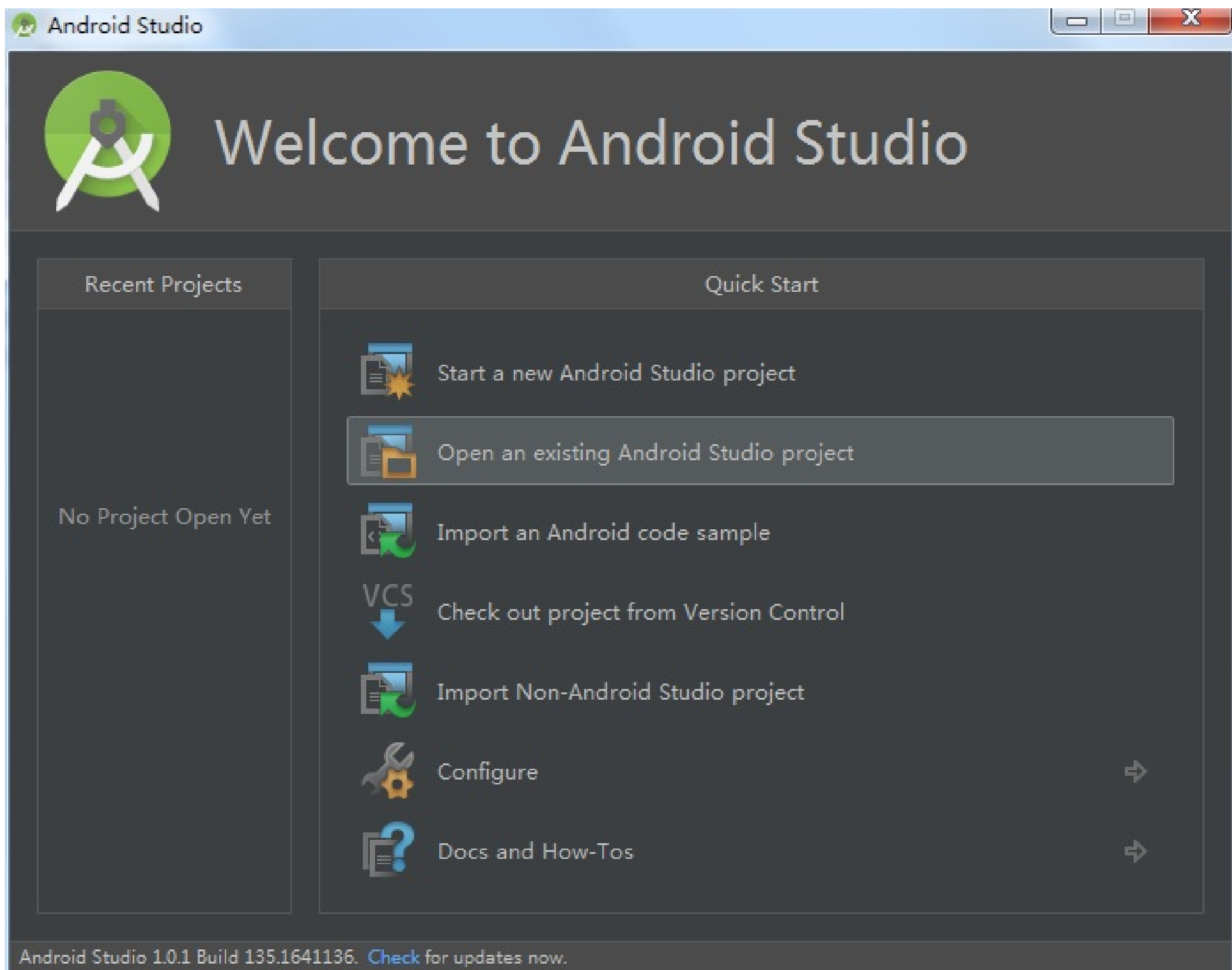
任务执行成功以后,在mylibrary\/build\/outputs\/aar目录下就会打出.aar格式的包.

默认Debug和Release的AAR包都会打出来,当然你也可以选择只打Debug的包,双击assembleDebug任务就可以了. 只打Release的包同理.

导入Android Studio项目

当你从网络上或者其他地方获取到一份Android Studio项目源码时，你希望能够导入到Android Studio中。首先，你需要先对这份源码进行一些修改。

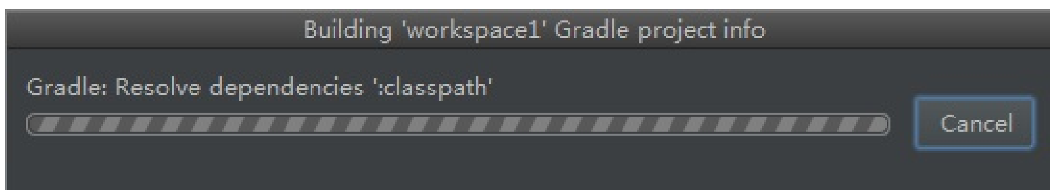
打开Android Studio，选择 **Open an existing Android Studio project**。



在弹出的路径选择框中，选择你要导入的Android Studio项目，点击 **OK** 。

讲解**23**

然后会弹出和 讲解**11** 相同的页面，同样，和 讲解**11** 进行一样的设置即可。

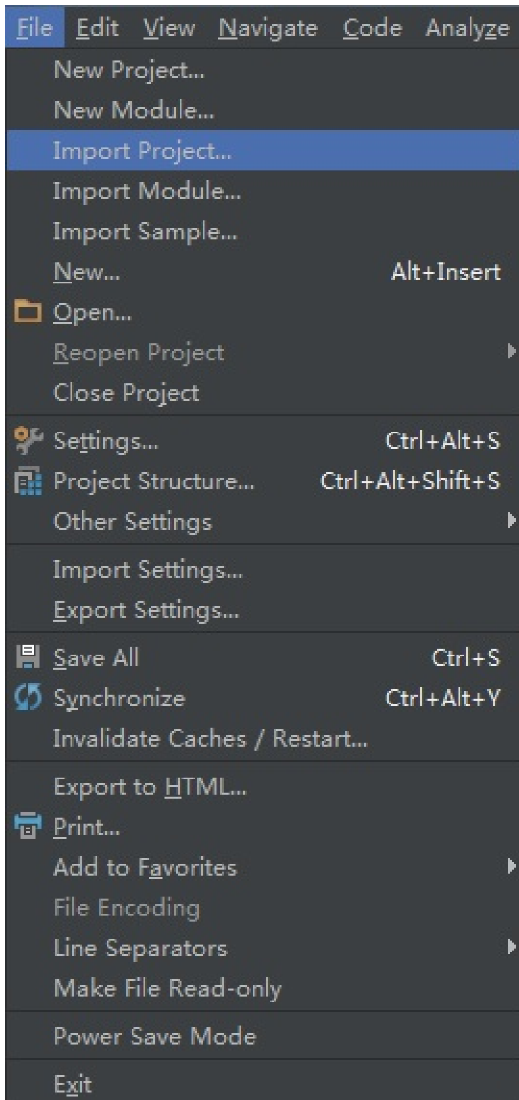


之后的步骤，同 讲解**11** ，在此不再赘述。

其他导入方式

导入一个项目除了在Android Studio起始页进行导入之外，你也可以在Android Studio主页中进行导入。

File --> Import Project 或者 **File --> Import Module** 。



导入的过程和前面的讲解都是一样的，就不在赘述了。

导入eclipse项目

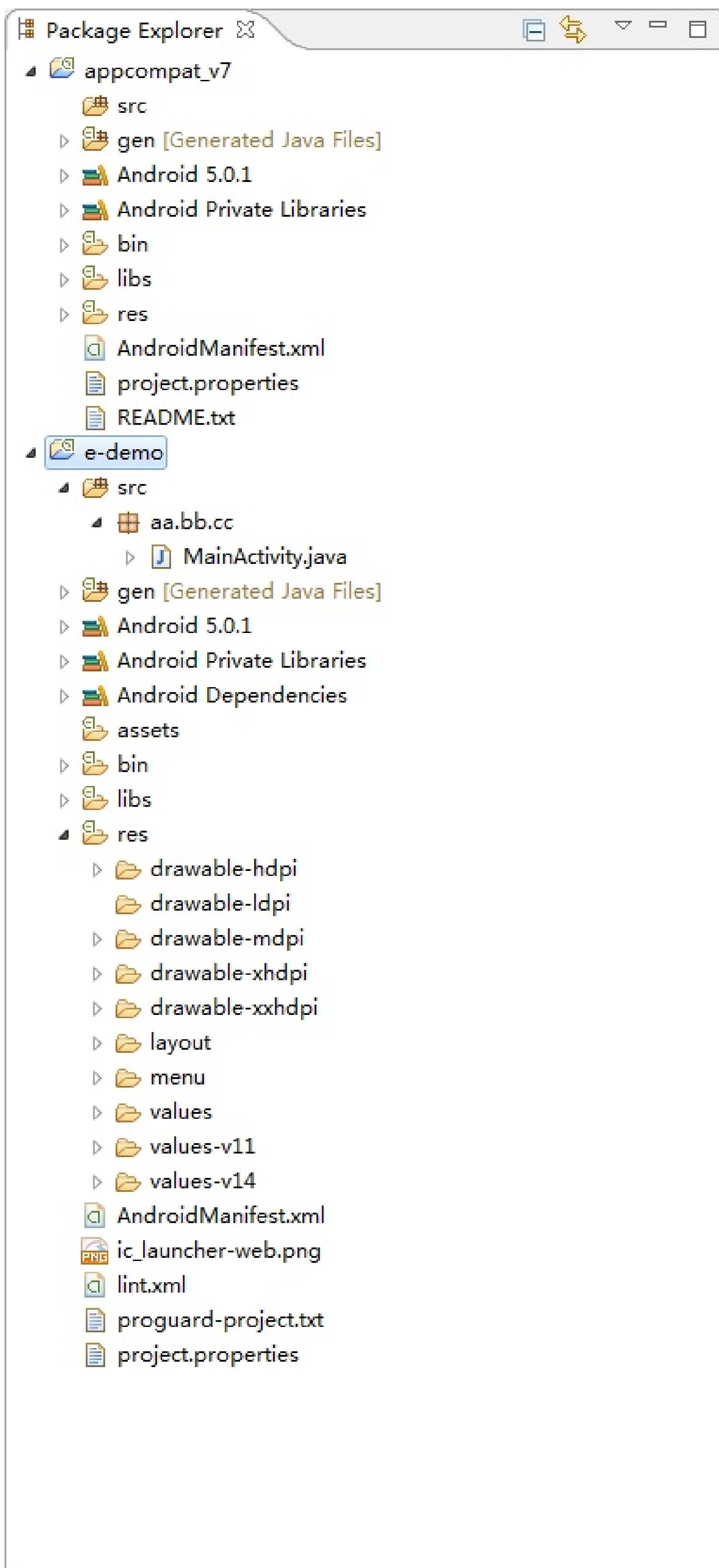
原文地址：http:\/\/ask.android-studio.org\/?\/article\/21

本篇教程中使用 Eclipse ADT版本23.0.4。请尝试更新到该版本。

Android Studio默认使用 **Gradle** 构建项目， **Eclipse** 默认使用**Ant**构建项目。建议Android Studio导入项目时，使用 **Gradle** 构建项目。

导入 Eclipse 项目

本例中，使用到的 **Eclipse** 项目结构如图：



e-demo 为主项目，**appcompat_v7** 为 **library** 项目。

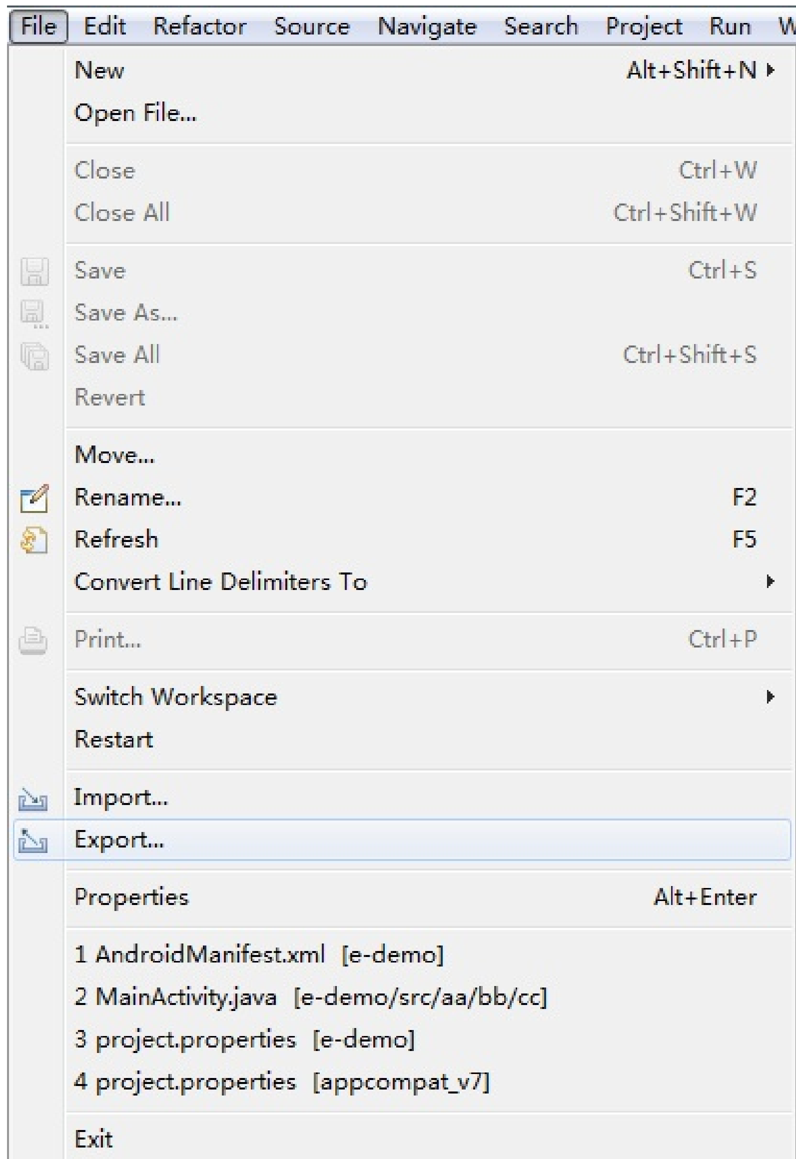
导入 Generate Gradle build files 项目

Google官方建议是通过本方法进行Android Studio导入 Eclipse 项目。

这种方式有一个好处就是兼容 Eclipse 的文件目录结构，通过版本控制中的文件过滤，可以在一个项目组中，同时使用 Eclipse 和Android Studio。

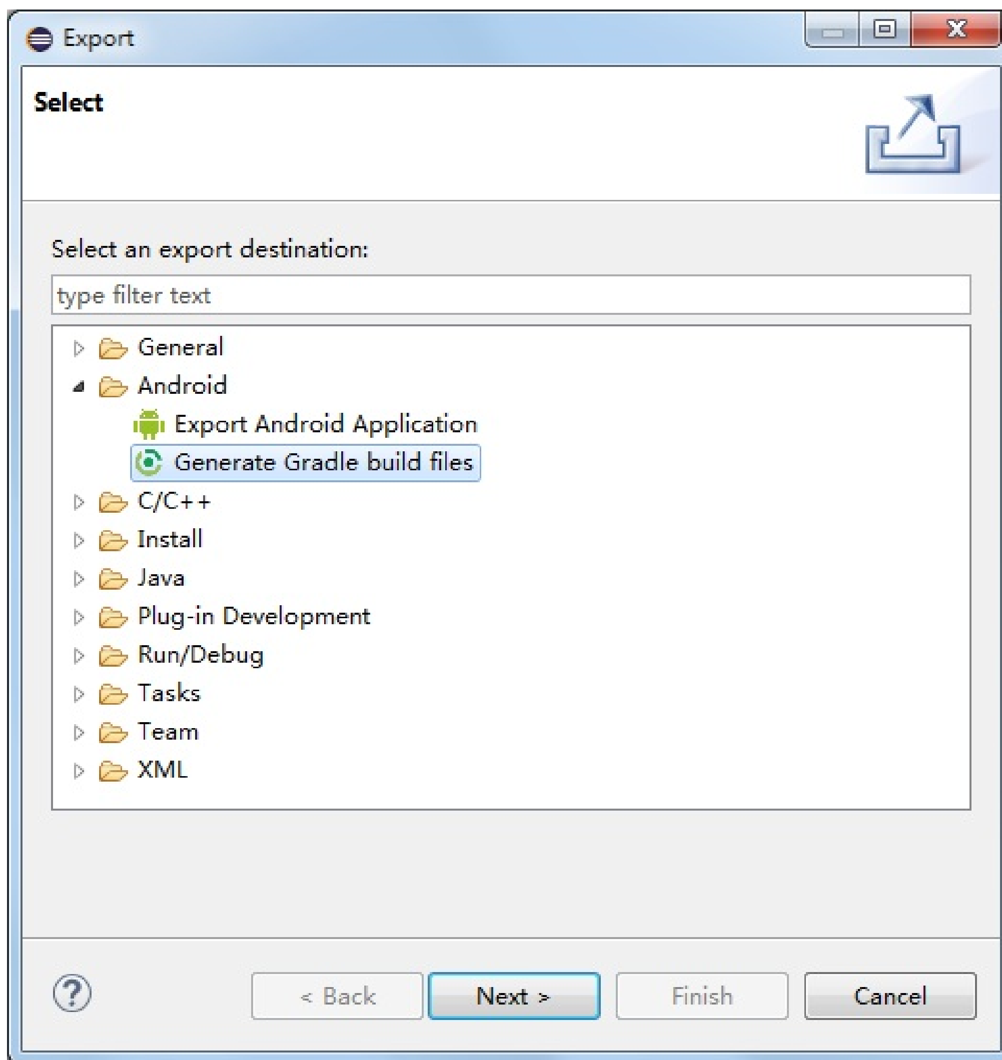
讲解1

File --> Export



讲解2

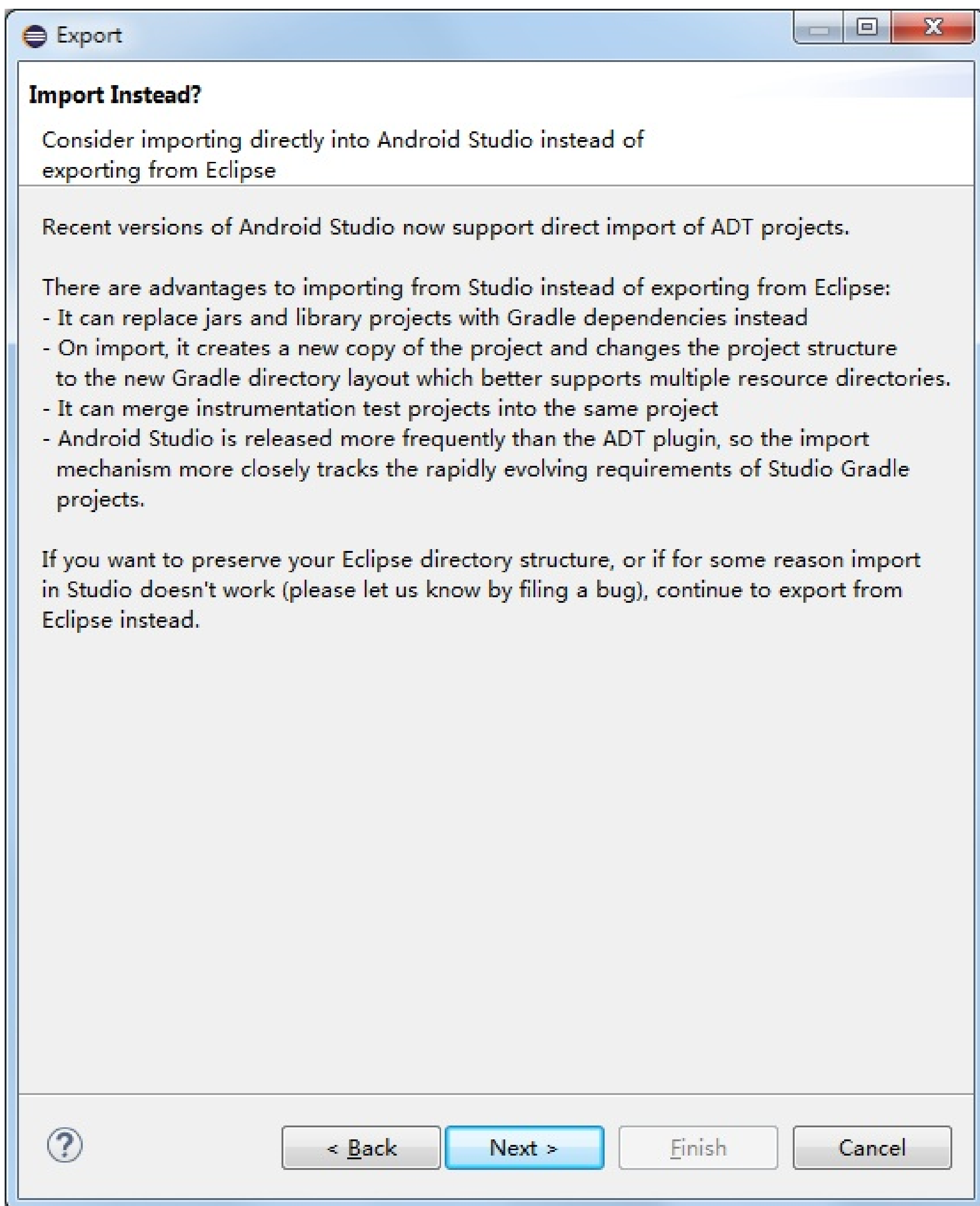
选择导出类型。选择 **Android --> Generate Gradle build files** 。



点击 **Next** 。

讲解**3**

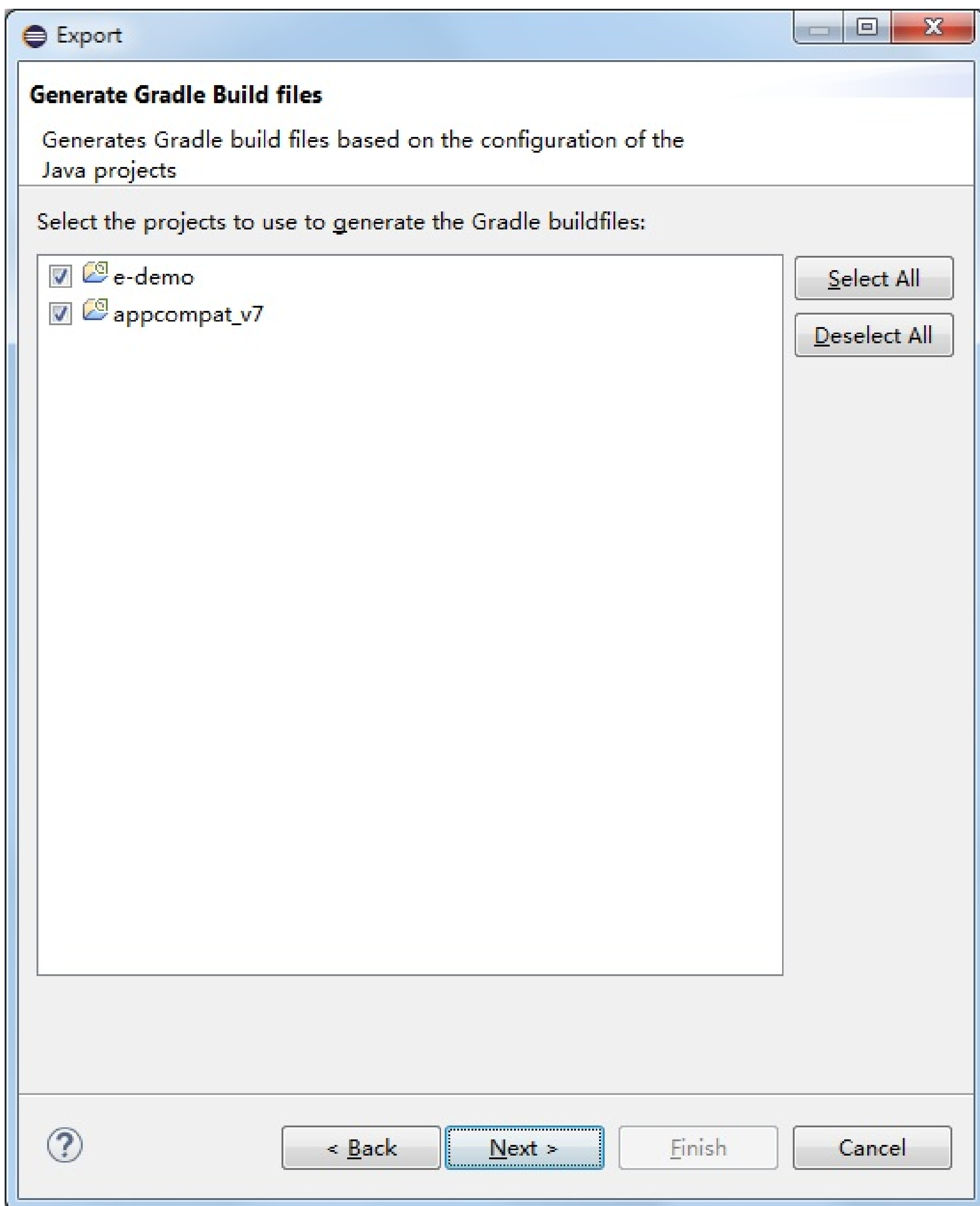
很长一段英语（完全看不懂是什么意思）。



点击 **Next** 。

讲解**4**

选择要导出的项目。

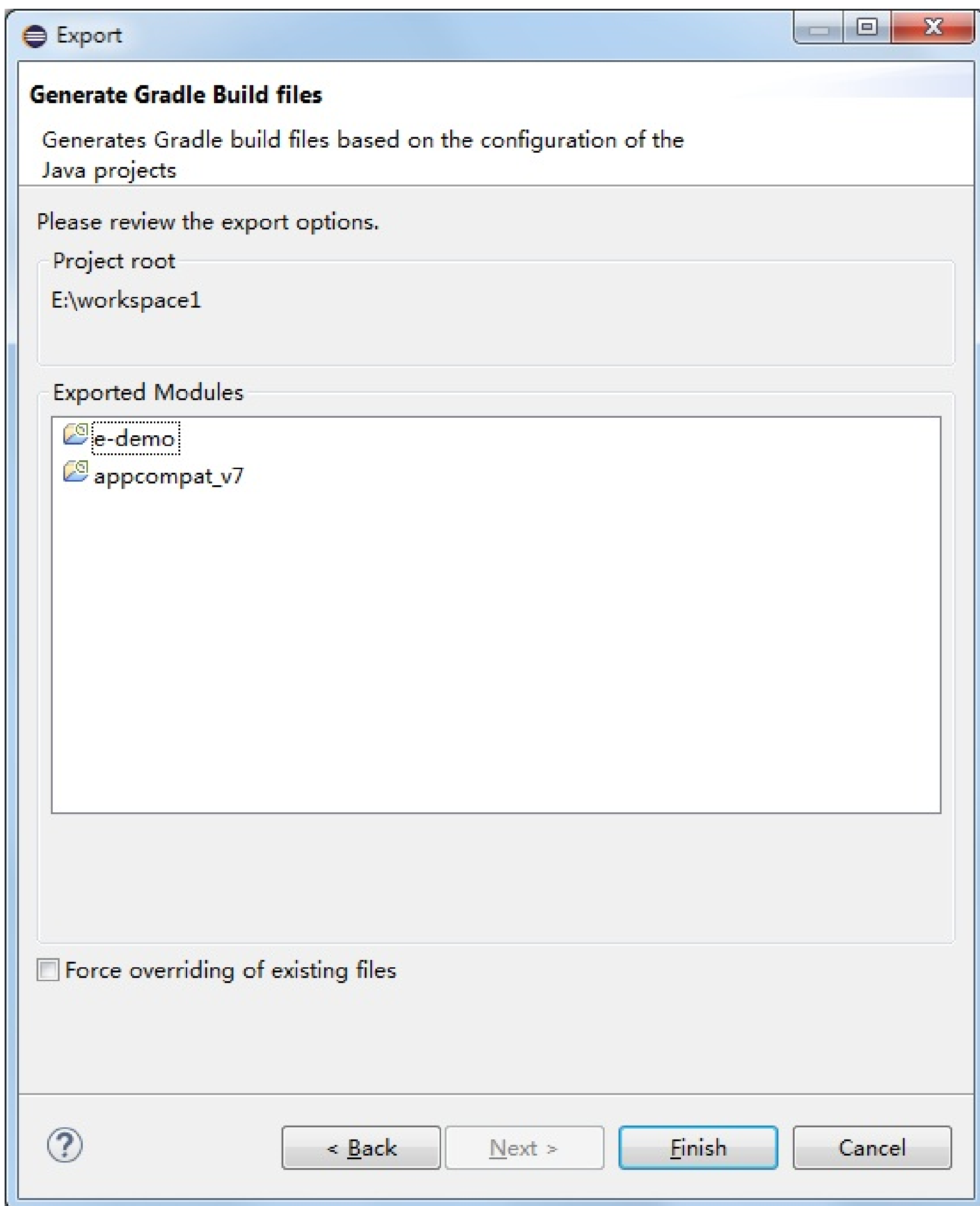


因为我的 **e-demo** 项目依赖了 **appcompat_v7** 项目，所以我将 **e-demo** 和 **appcompat_v7** 都选择了导出。

点击 **Next** 。

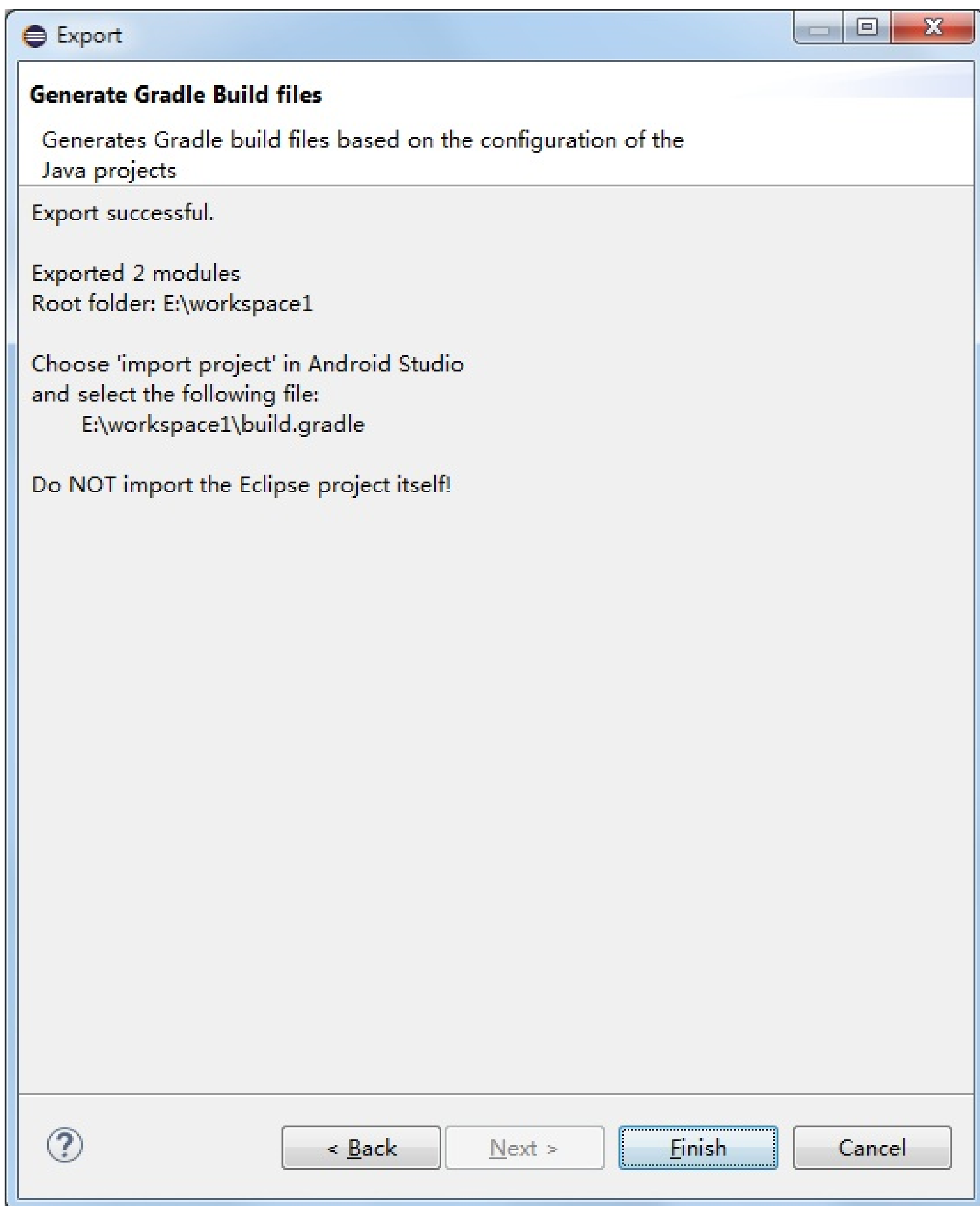
讲解**5**

最终确认要导出的项目。



Force overriding of existing files 表示覆盖导出文件。使用 **Generate Gradle build files** 的方式导出项目，会在项目目录中生成一些文件。这里的覆盖文件指的就是覆盖这些可能已经生成过的文件。如果你之前有使用这种方式导出过项目，建议勾选。

点击 **Finish** 。



讲解6

这一步没有什么好说的，直接点击 **Finish** 。

讲解7

Finish 点击完毕，并没有弹出窗口显示导出的项目，就好像什么事情都没有做一样。其实，使用这个方式导出项目，是在项目中添加了一些文件，我们可以到项目目录下去看一看这些生成文件。

工作空间目录下

文档 (E:) ▸ workspace1 ▸

共享 ▾ 新建文件夹

名称	修改日期	类型	大小
.metadata	2014/12/16 周二...	文件夹	
appcompat_v7	2014/12/16 周二...	文件夹	
e-demo	2014/12/16 周二...	文件夹	
gradle	2014/12/16 周二...	文件夹	
build.gradle	2014/12/16 周二...	GRADLE 文件	1 KB
gradlew	2014/12/16 周二...	文件	5 KB
gradlew.bat	2014/12/16 周二...	Windows 批处理...	3 KB
settings.gradle	2014/12/16 周二...	GRADLE 文件	1 KB

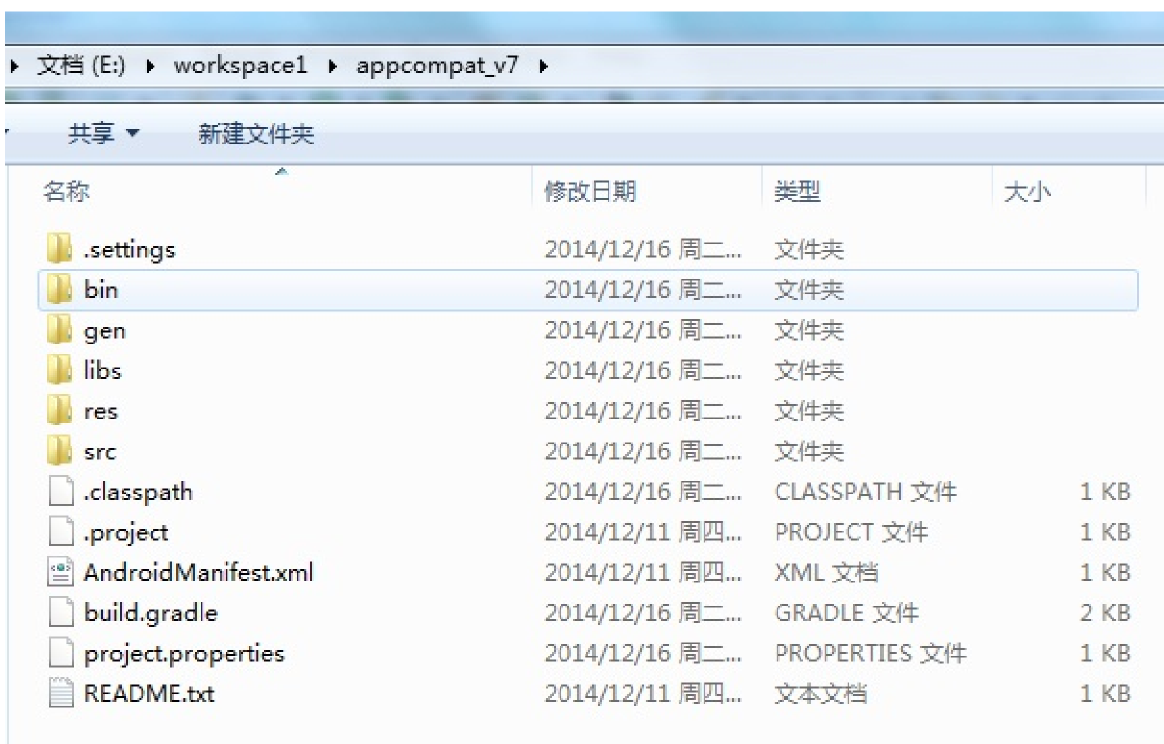
e-demo 目录下

文档 (E:) ▸ workspace1 ▸ e-demo ▸

共享 ▾ 新建文件夹

名称	修改日期	类型	大小
.settings	2014/12/16 周二...	文件夹	
assets	2014/12/16 周二...	文件夹	
bin	2014/12/16 周二...	文件夹	
gen	2014/12/16 周二...	文件夹	
libs	2014/12/16 周二...	文件夹	
res	2014/12/16 周二...	文件夹	
src	2014/12/16 周二...	文件夹	
.classpath	2014/12/16 周二...	CLASSPATH 文件	1 KB
.project	2014/12/16 周二...	PROJECT 文件	1 KB
AndroidManifest.xml	2014/12/16 周二...	XML 文档	1 KB
build.gradle	2014/12/16 周二...	GRADLE 文件	2 KB
ic_launcher-web.png	2014/12/16 周二...	PNG 文件	51 KB
lint.xml	2014/12/16 周二...	XML 文档	1 KB
proguard-project.txt	2014/12/16 周二...	文本文档	1 KB
project.properties	2014/12/16 周二...	PROPERTIES 文件	1 KB

appcompat_v7 目录下



我们可以发现：在工作空间目录下，多出了 **gradle** 文件夹和 **build.gradle** 、 **build.gradle** 、 **gradlew** 、**gradlew.bat** 、 **settings.gradle** 文件；在 **e-demo** 目录下多出了 **build.gradle** 文件； 在 **appcompat_v7** 目录下多出了 **build.gradle** 文件。这些文件和文件夹都和 **Gradle** 有关系，用于构建项目。这些文件以及文件夹的作用，我们以后再说。

讲解**8**

由于 **Eclipse** 的 **ADT** 插件已经很久没有更新了，自动生成的 **Gradle** 编译设置已经跟不上 Android Studio的更新速度，所以我们需要手动修改一些内容。

打开工作空间目录下的 **gradle --> wrapper --> gradle-wrapper.properties** 。修改一下内容：**distributionUrl=http\:\/\/services.gradle.org\/distributions\/gradle-a.b.c-all.zip -->distributionUrl=https\:\/\/services.gradle.org\/distributions\/gradle-2.2.1-all.zip**

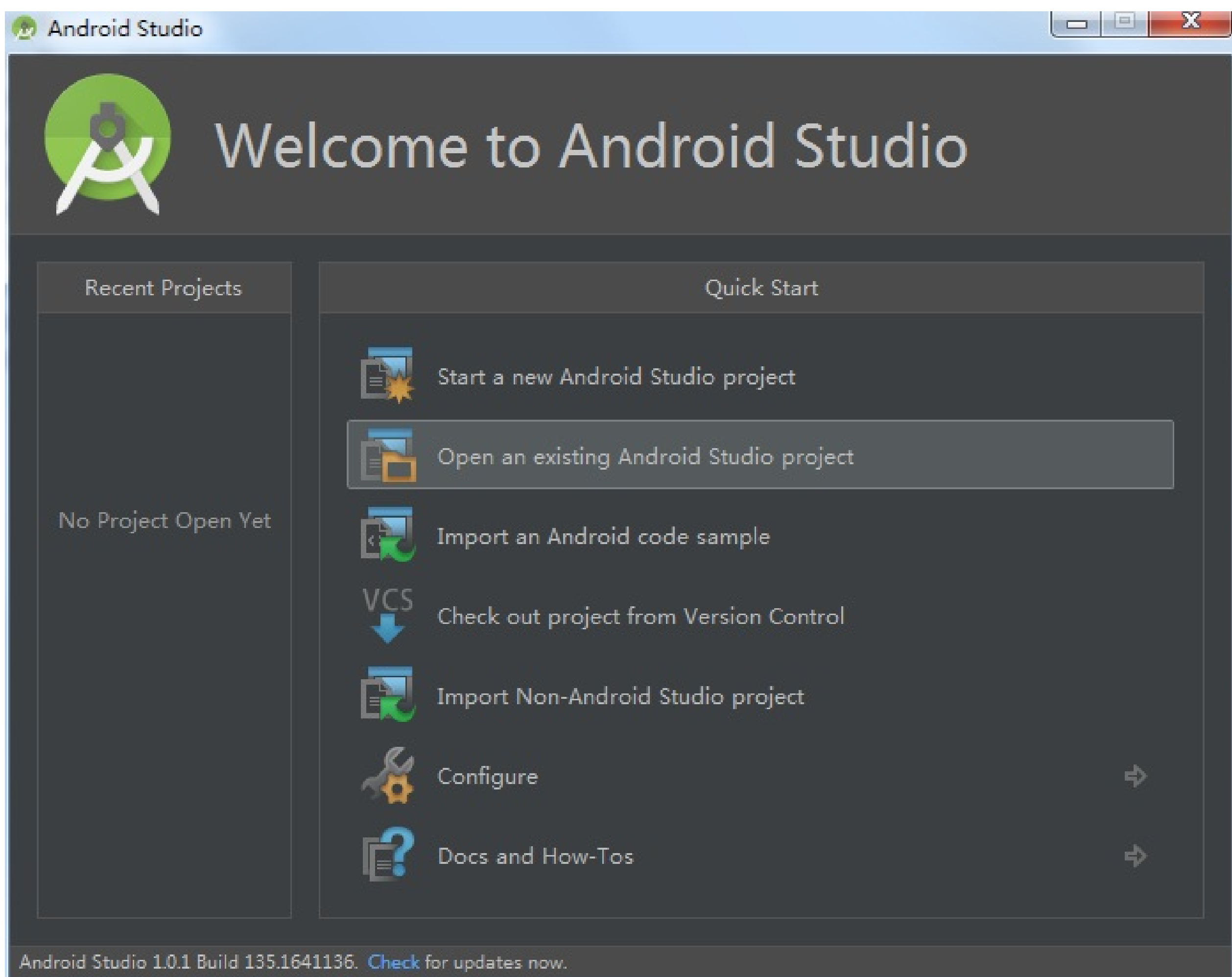
打开工作空间目录下的 **build.gradle** 文件。修改以下内容：

classpath 'com.android.tools.build:gradle:0.x.+' --> classpath 'com.android.tools.build:gradle:1.0.0' 。

之所以这么设置，是因为： **Eclipse** 导出的 **Gradle** 设置已经不是Android Studio 1.0 所支持的 **Gradle** 已经 **Gradle**插件版本，需要手动更为支持的版本。否则轻则必须不能离线导入项目，重则项目导入失败。

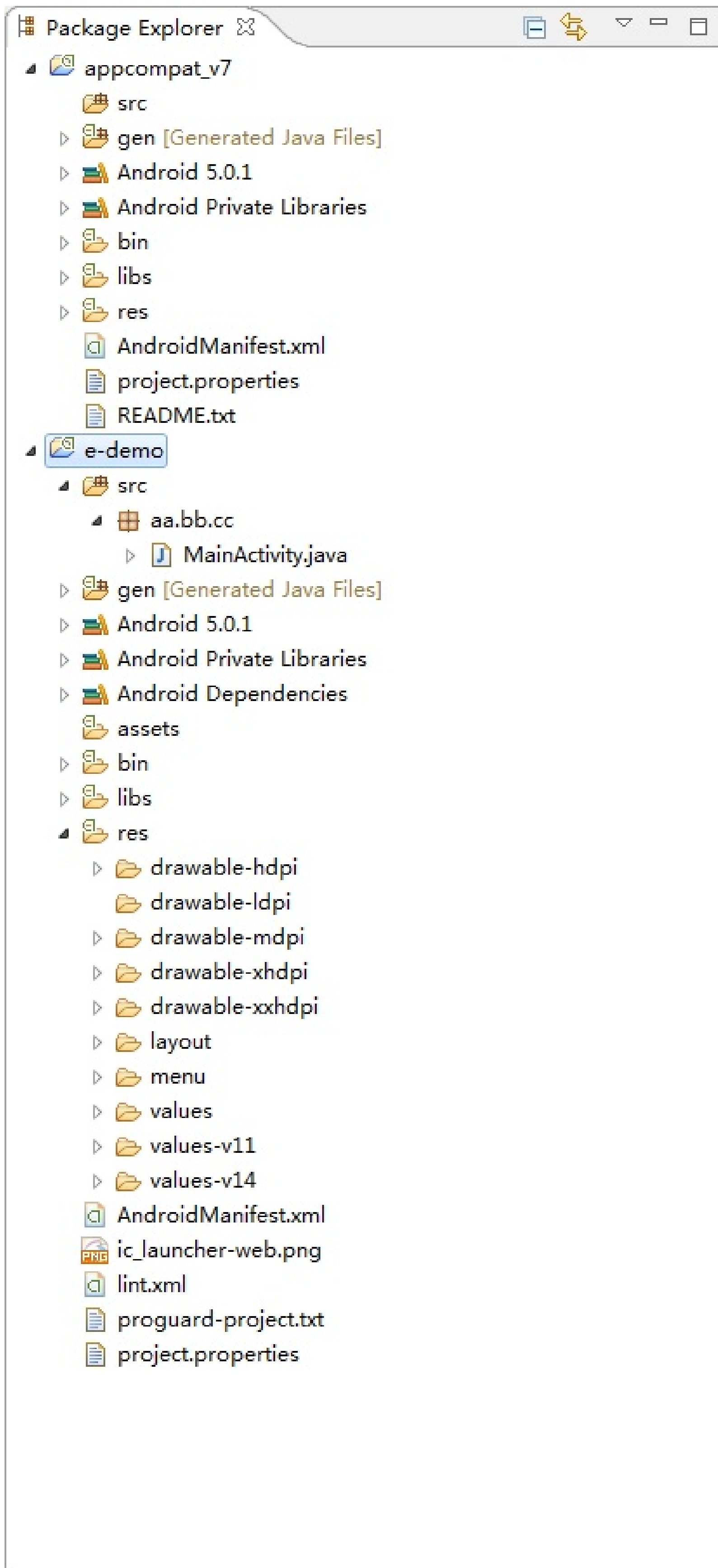
讲解**9**

打开Android Studio，选择 **Open an existing Android Studio project**。



讲解**10**

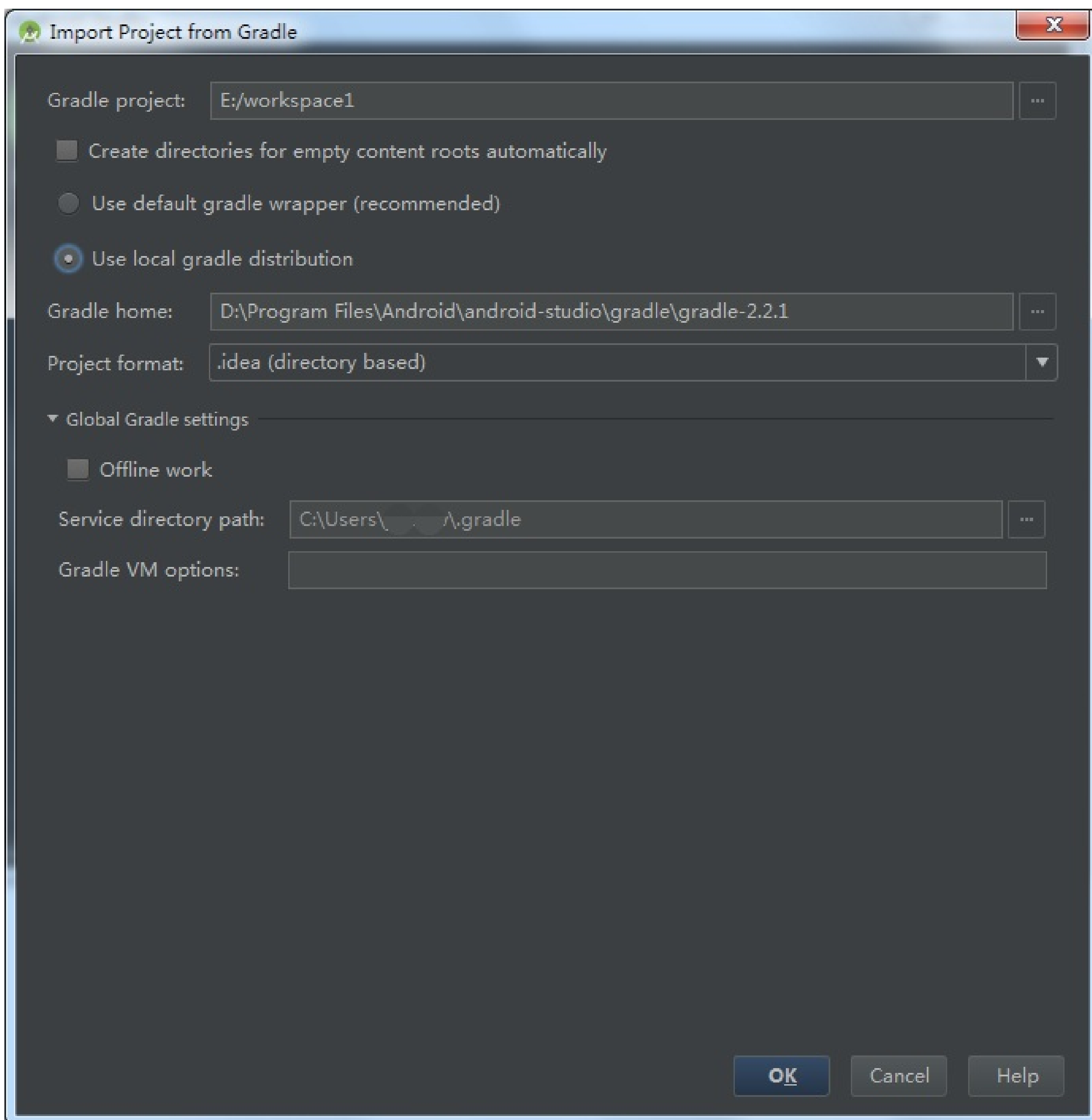
此时会弹出一个框，让你选择文件夹，这个时候需要注意的就是，你需要选择原来的 **Eclipse** 的工作空间目录，而不是 **e-demo** 目录。


Package Explorer
appcompat_v7
src
gen [Generated Java Files]
Android 5.0.1
Android Private Libraries
bin
libs
res
AndroidManifest.xml
project.properties
README.txt
e-demo
src
aa.bb.cc
MainActivity.java
gen [Generated Java Files]
Android 5.0.1
Android Private Libraries
Android Dependencies
assets
bin
libs
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
menu
values
values-v11
values-v14
AndroidManifest.xml
ic_launcher-web.png
lint.xml
proguard-project.txt
project.properties

点击 **OK** 。

讲解11

设置导入选项。



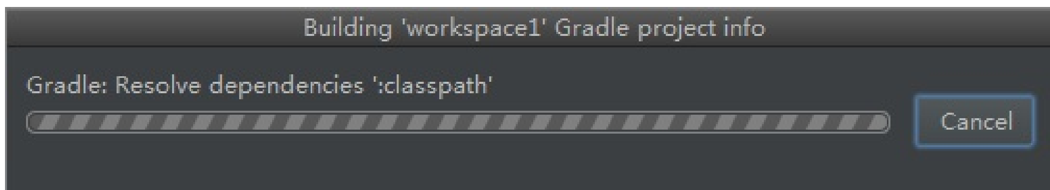
此处有一些比较重要的设置需要讲解一下。

- **Gradle project** ：此处通常显示的路径并不是你的 **Eclipse** 的工作空间的目录，而是 **Eclipse** 的工作空间的目录中的 **gradle** 路径。你需要手动删除后面的 **gradle** ，否则项目导入，你是看不到你的代码的，只能看到**gradle** 目录下的内容。
- **Create directories for empty content roots automatically** ：不是很明白它的作用，一般默认即可。
- **Use default gradle wrapper(recommended)** 和 **Use local gradle disribution** ：这两个是让你设置使用的**Gradle** 。默认会勾选 **Use default gradle wrapper(recommended)** ，我们需要手动勾选 **Use local gradle disribution** 。
- **Gradle home** ：勾选 **Use local gradle disribution** 后此项编程可编辑状态，默认的此处的地址为Android Studio安装目录中的 **Gradle** 路径地址。此处可能会有一些错误的警

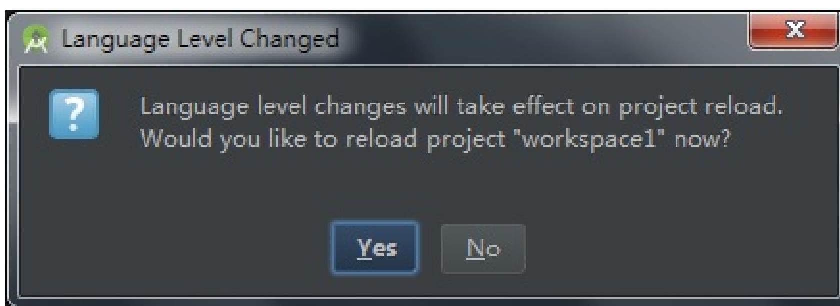
告，提示内容为： **Gradle location is incorrect** 。而你的这个目录下，确实是有 **Gradle** 的。产生这个问题的原因，很可能是因为 **Gradle home** 选项中，路径中的斜杠为 \/ 而不是 ** 。你需要点击左右的文件选择按钮，重新选择到**Android Studio**安装目录中的 **Gradle ，问题即可解决。

- **Offline work** ：设置 **Gradle** 使用离线的方式导入项目。你可以勾选也可以不勾选。如果你有进行 讲解**8** 的操作，你则可以勾选，以离线的方式进行编译。

点击 **OK** 。之后便会看到编译进度条，根据每个人机器的配置，编译的时间不同。

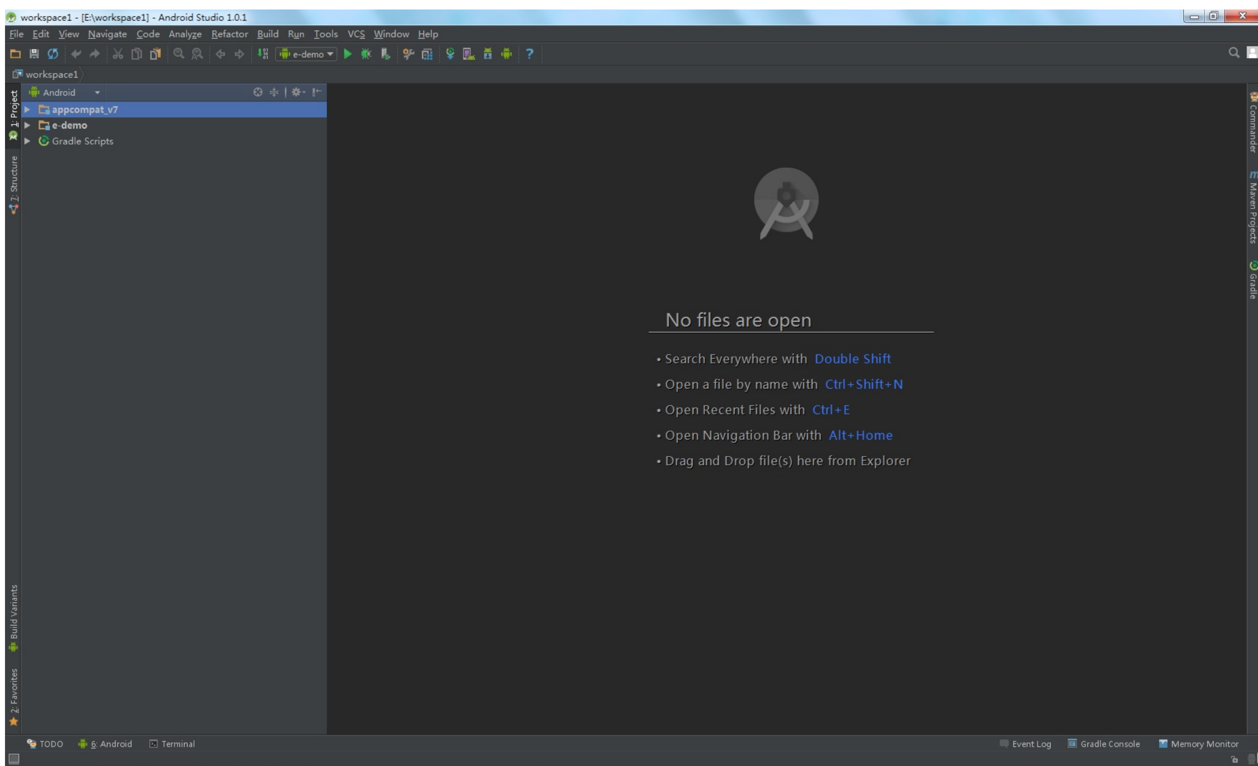


编译完成之后，自动跳转到Android Studio的主页面。在编译的工程中，会有以下的弹框：



之所以有这个弹框，是因为Android Studio默认使用 **JAVA 1.7** 进行编译，如果你的项目不是 **1.7** ，则会弹框让你选择。建议选择 **Yes** ，因为当你使用 **JAVA 1.7** 的时候，只要不使用 **JAVA 1.7** 的资源自动释放这个新特性，能够完美得兼容 **JAVA 1.6** 的Android设备。

如果你看到下面这个界面，说明你已经导入成功了。



直接导入 **Eclipse** 项目

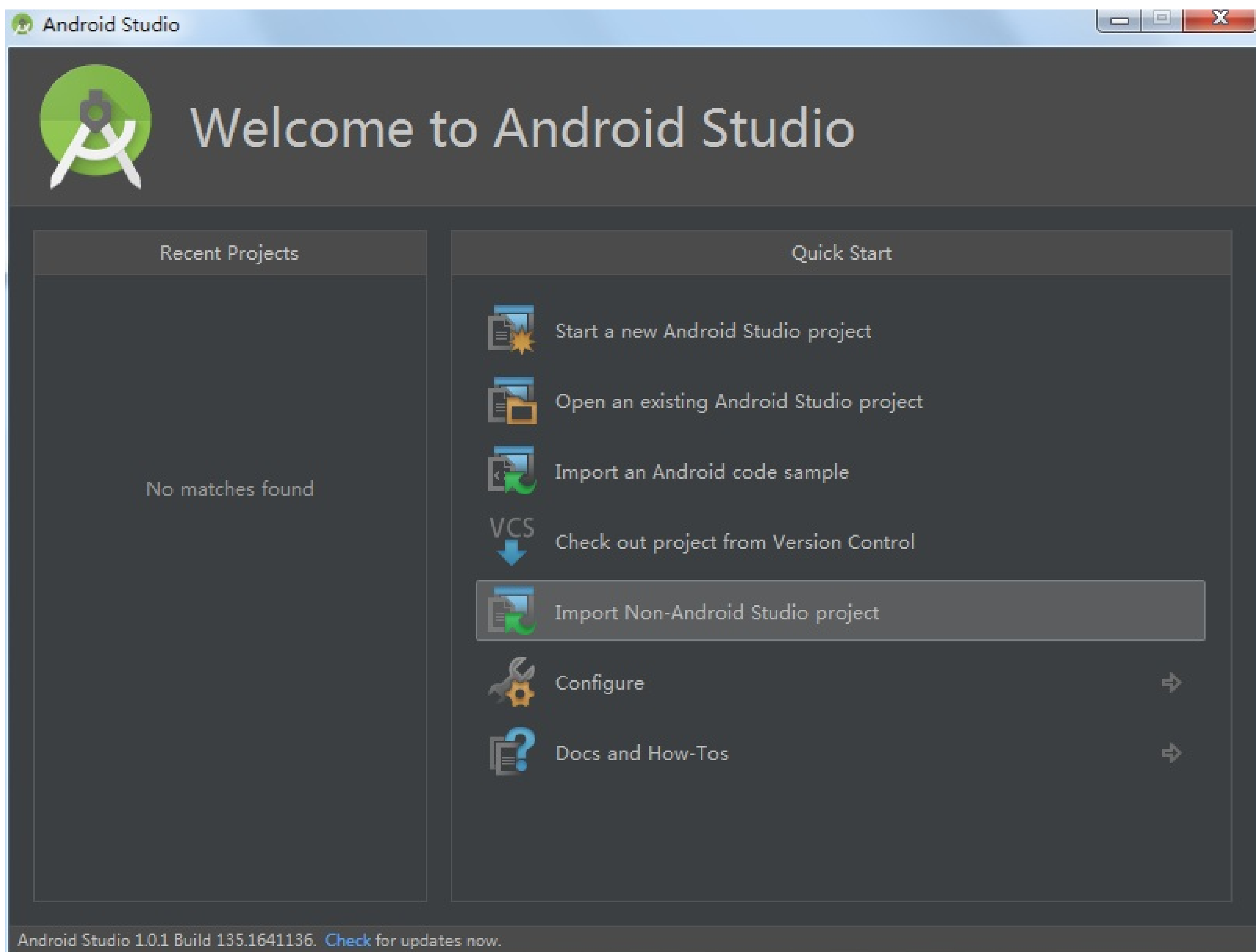
如果不使用 **Generate Gradle build files** 导出项目，可以使用Android Studio直接打开 **Eclipse** 工作空间，进行项目导入。

不使用 **Gradle** 编译项目

这种方式可以兼容 `Eclipse` 的文件目录结构，通过版本控制中的文件过滤，可以在一个项目组中，同时使用 **Eclipse** 和Android Studio。但是在Android Studio中并不是使用 **Gradle** 构建项目，而是使用的 **Ant** 。

讲解**12**

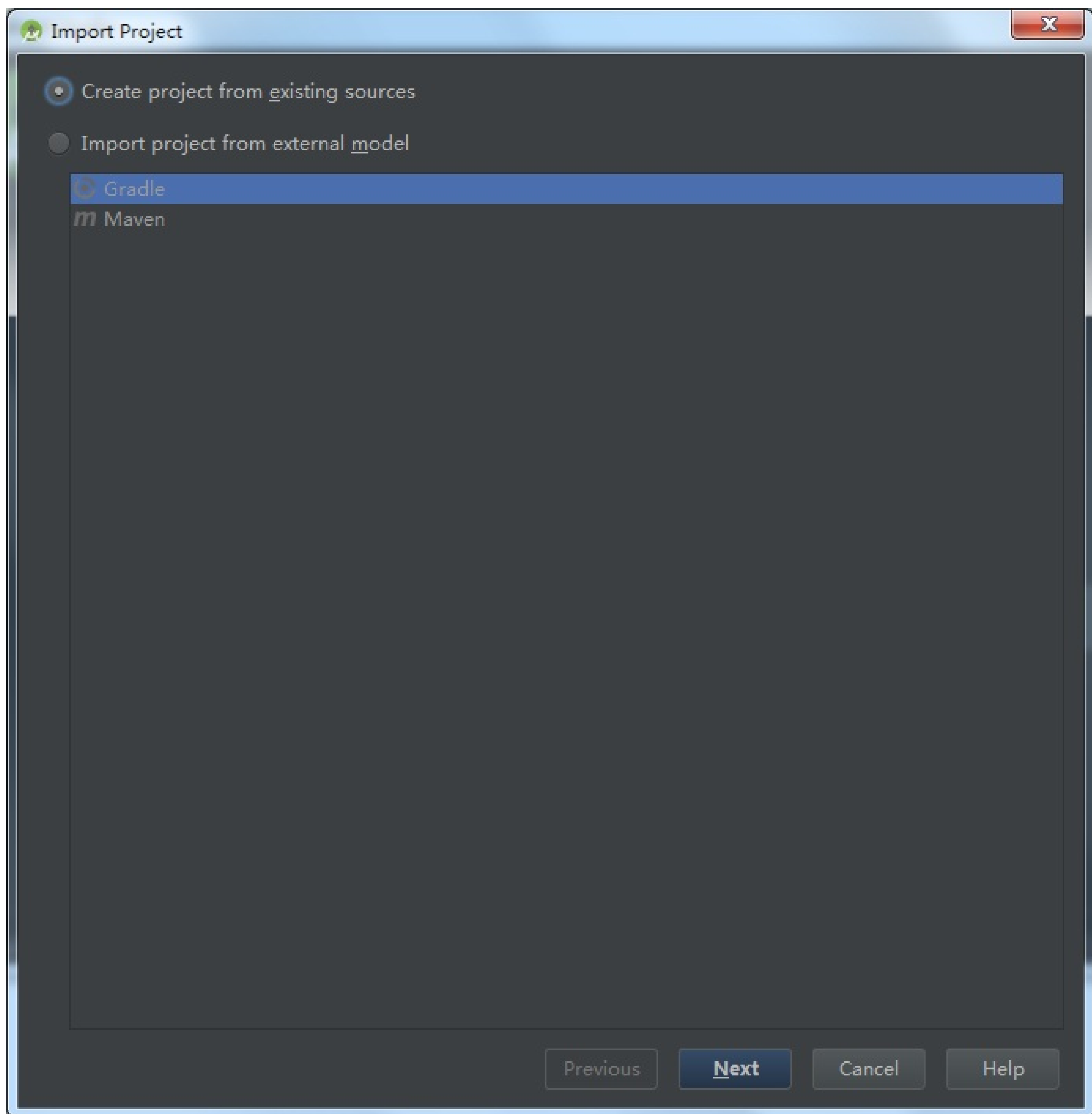
打开Android Studio，选择 **Import Non-Android Studio project**。



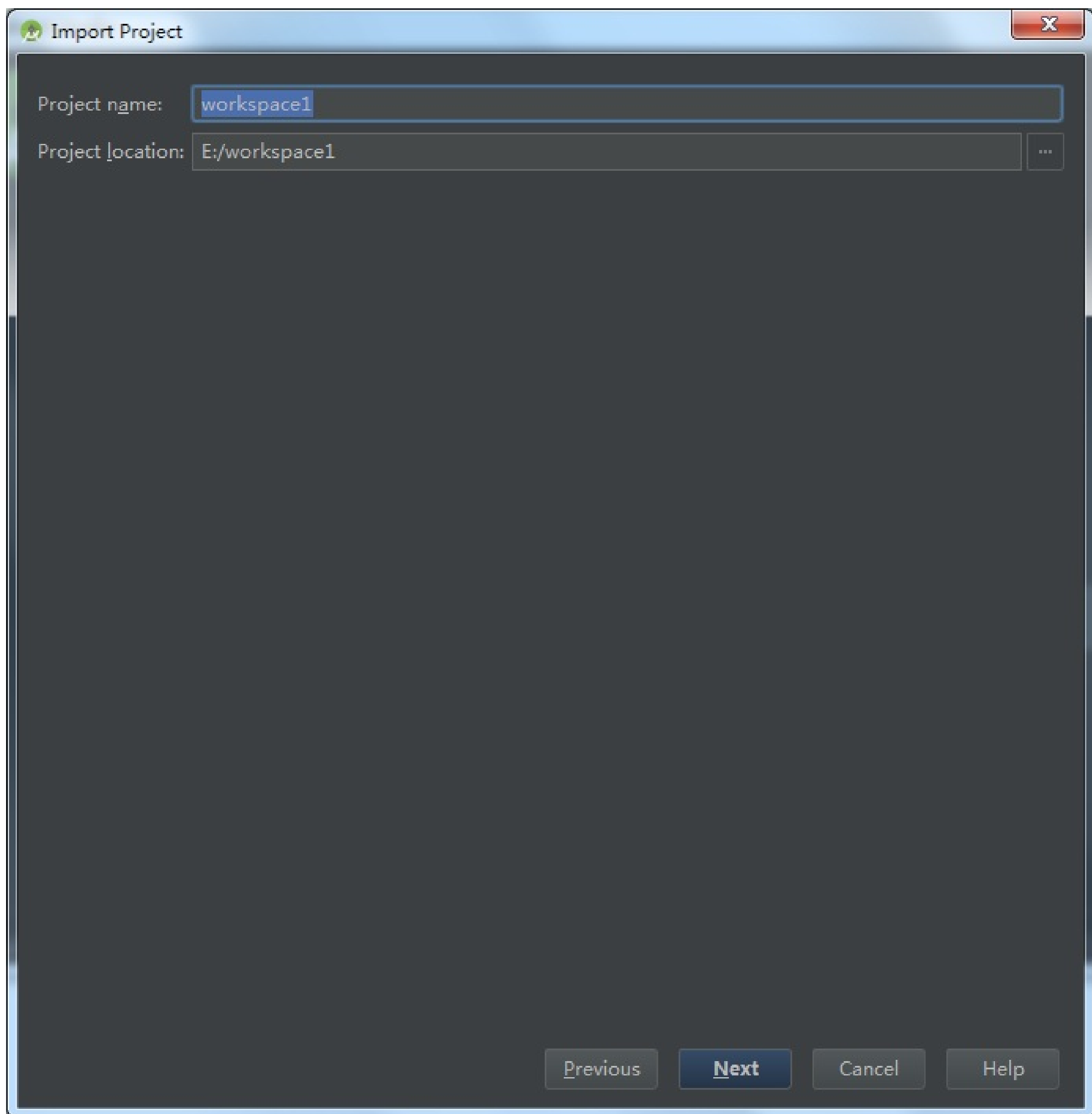
在弹出的目录选择框中，选择 **Eclipse** 的工作空间 。

讲解**13**

接下来回让你选择编译环境。选择 **Create project from existing sources** 。


Import Project
Create project from existing sources
Import project from external model
Gradle
Maven
Previous
Next
Cancel
Help

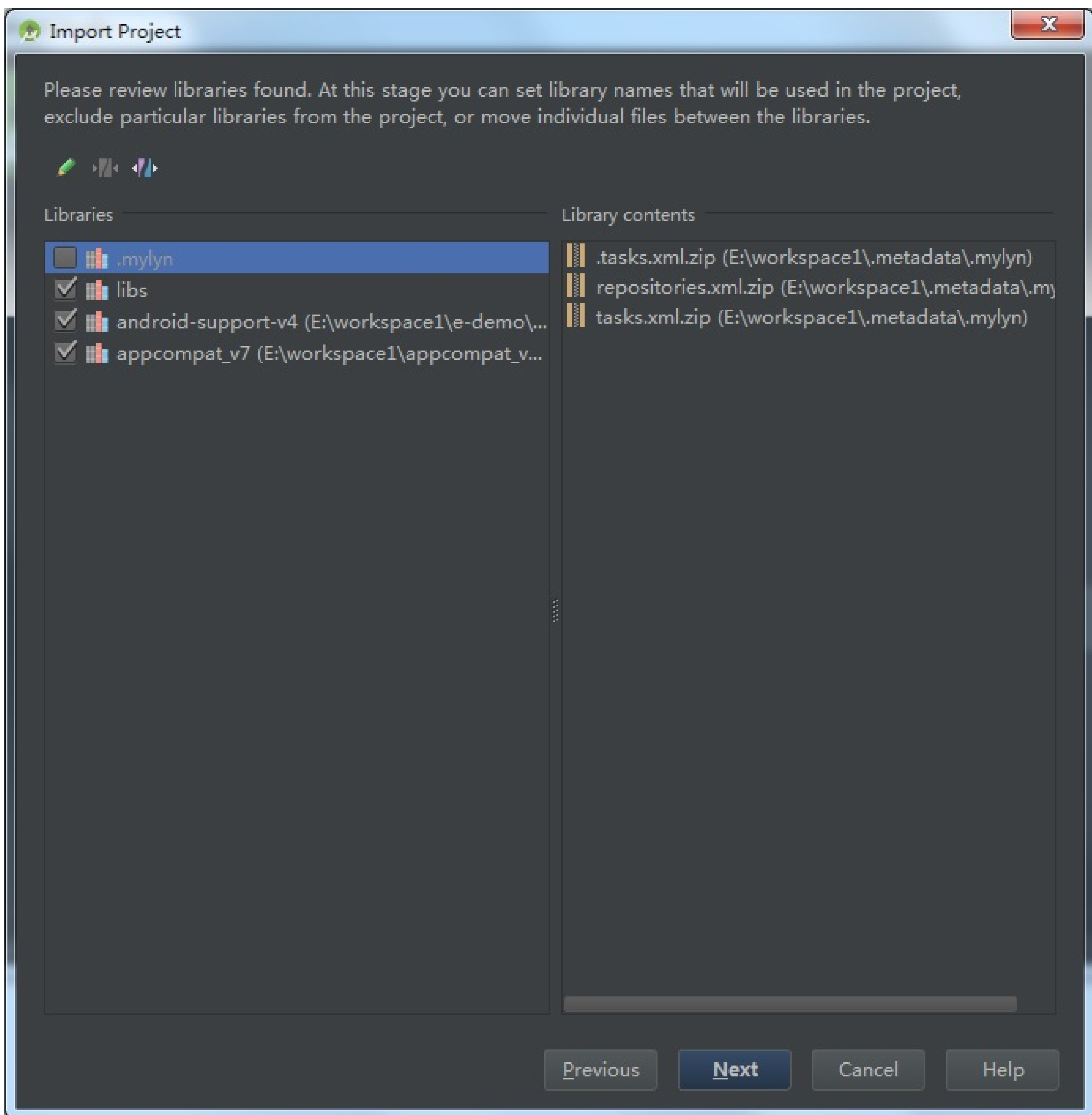
讲解**14**



设置 Android Studio **Project** 名称以及存放目录。一般默认即可。

点击 **Next** 。

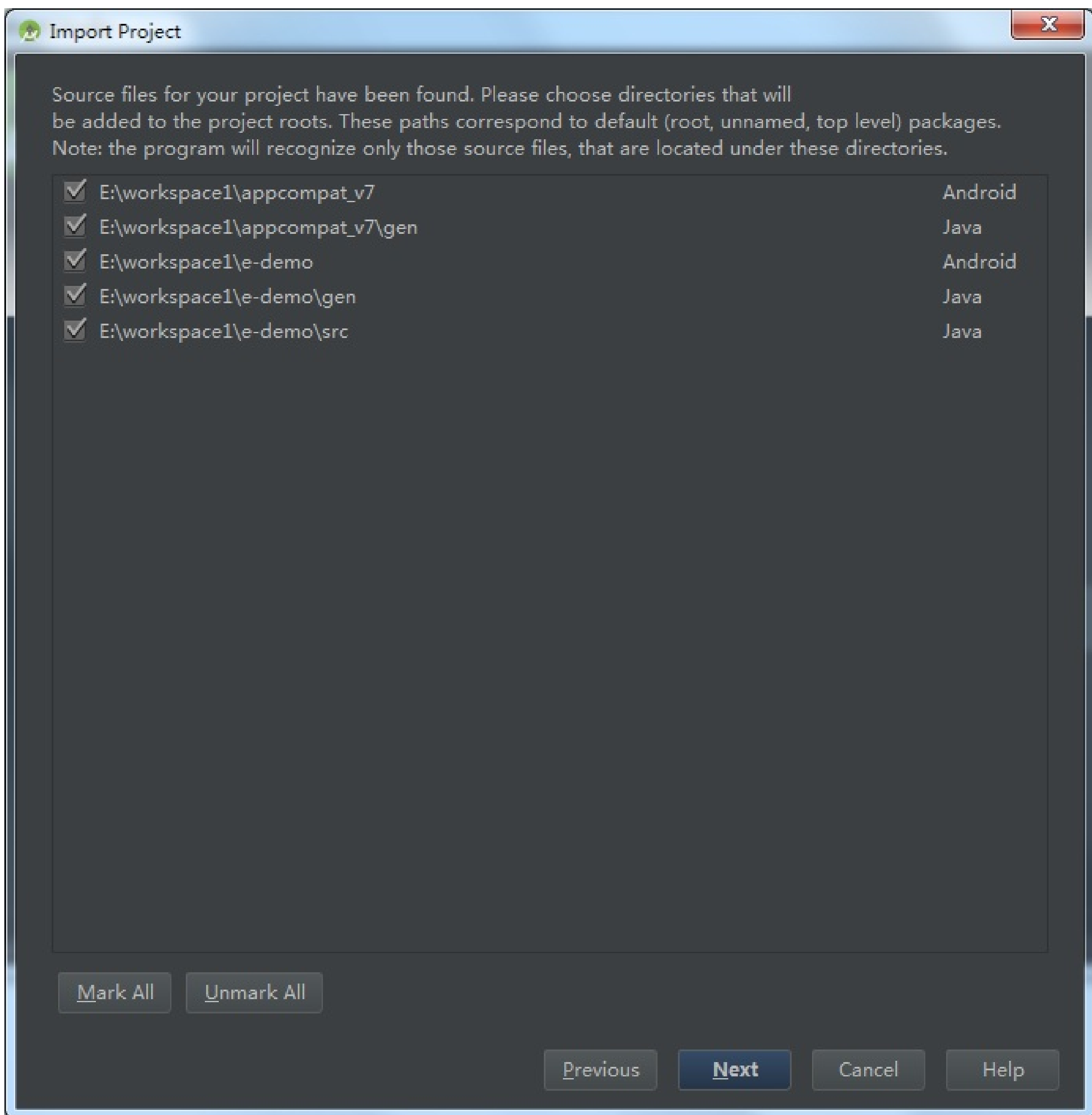
讲解**15**



选择资源文件以及资源文件夹。根据你需要导入的项目进行勾选。选择一个项目的时候，你需要选择它依赖的**Library** 项目以及他的 **src** 和 **gen** 目录。

选择完毕，点击 **Next** 。

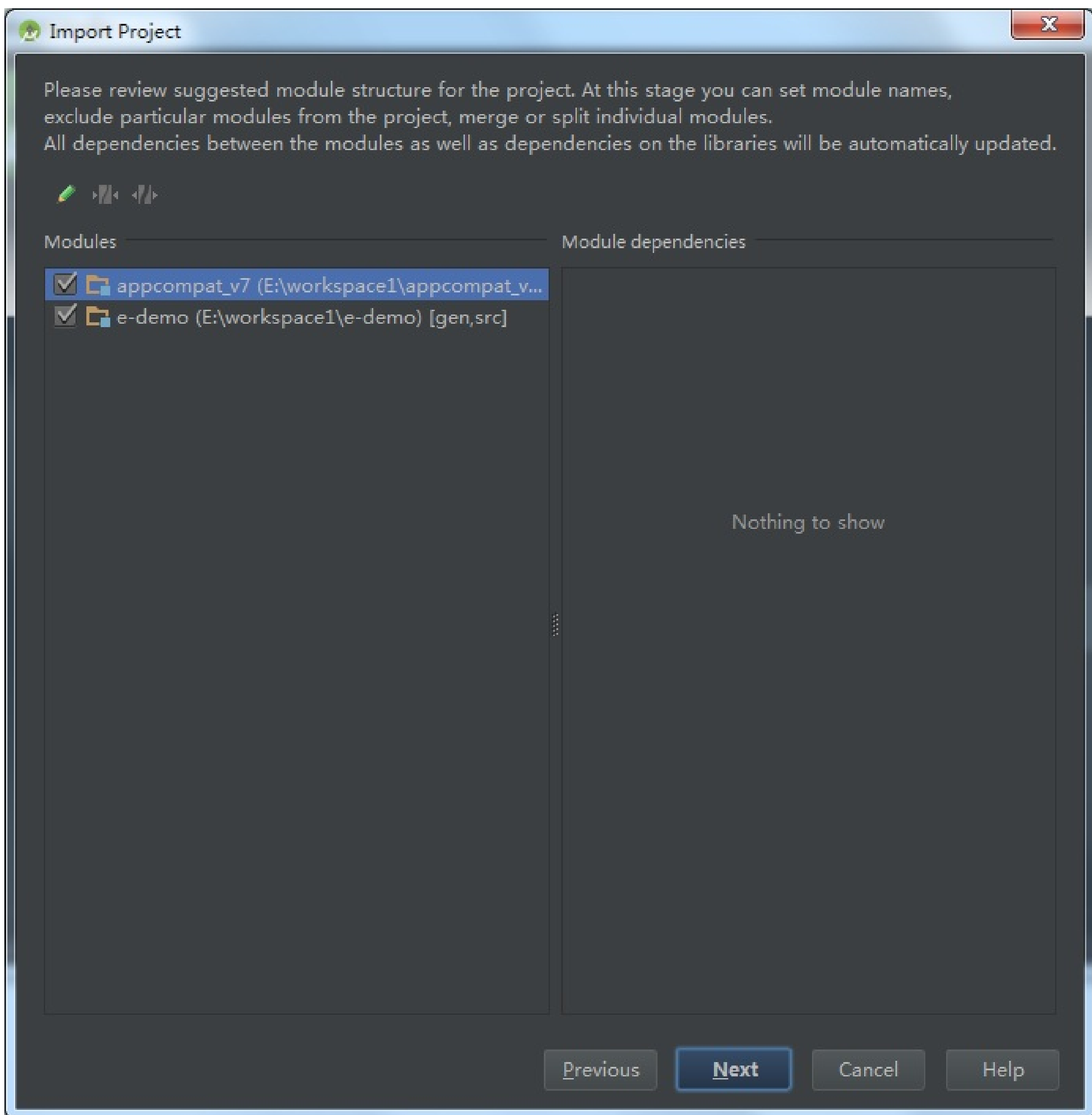
讲解16



选择要导入的 **jar** 。第一个 **.mylyn** 是 **Eclipse** 插件的内容，我们需要手动过滤掉，其他的根据你的需要，进行选择。

选择完毕，点击 **Next** 。

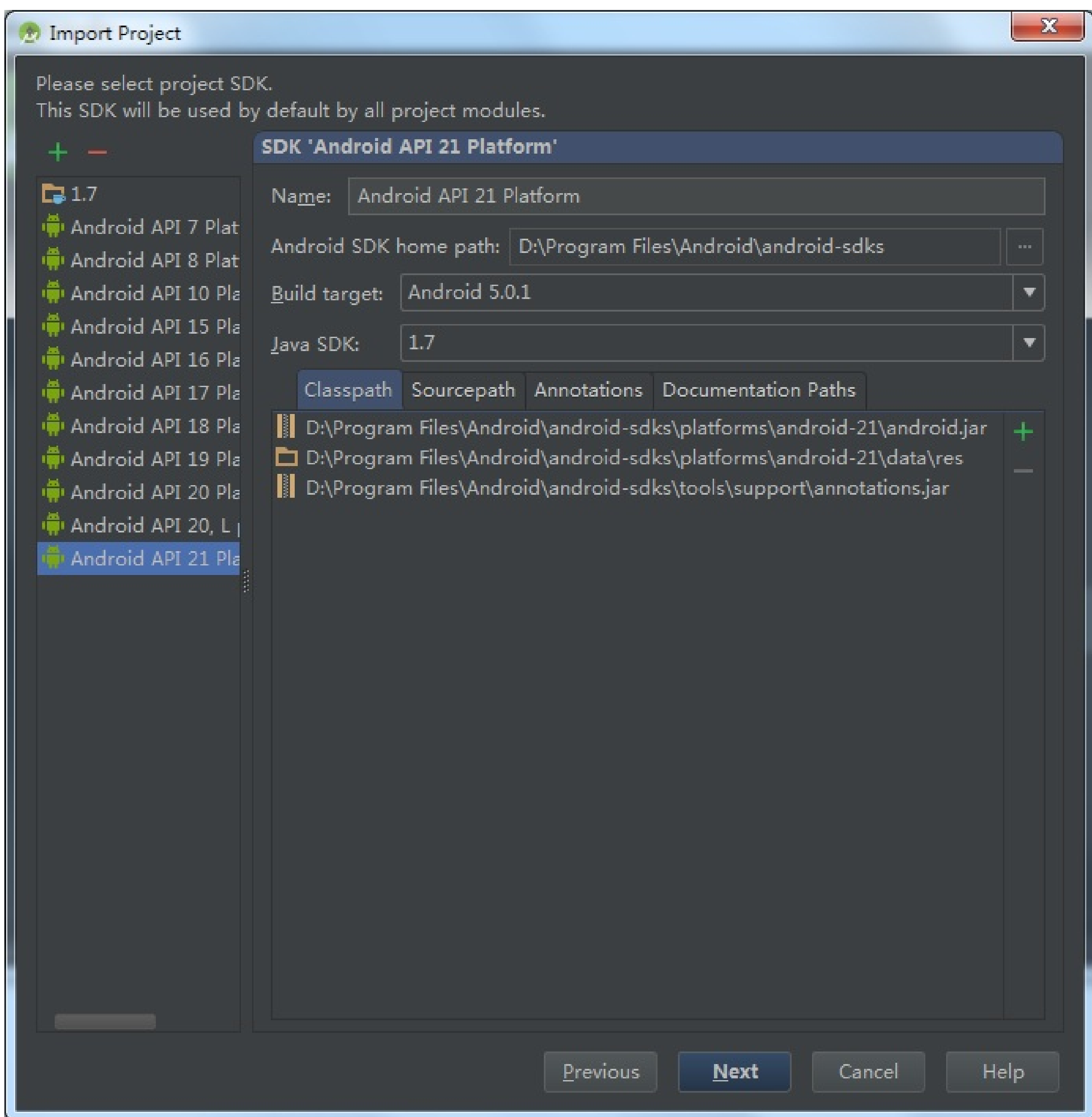
讲解**17**



选择Android Studio识别出的 **Module** ，也就是 **Eclipse** 中的 **Project** 。勾选你想导入的项目即可。

选择完毕，点击 **Next** 。

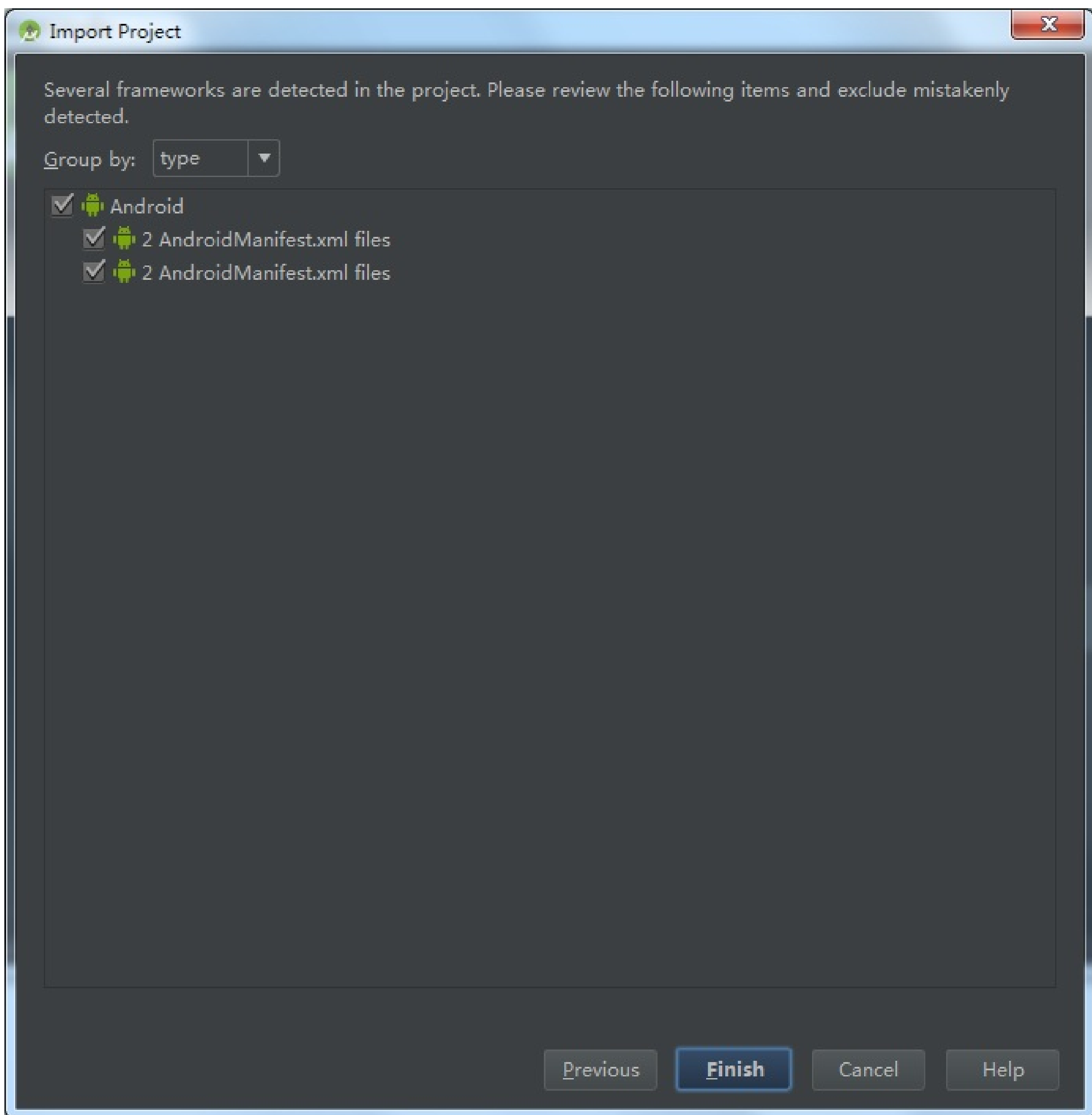
讲解**18**



选择编译的 **SDK** 。在这里，你需要选择 **Android SDK** ，最好和之前使用 **Eclipse** 时使用的 **SDK**一样。同样，你也可以再次进行一些简单的环境设置，在此就不说了。

选择完毕，点击 **Next** 。

讲解**19**

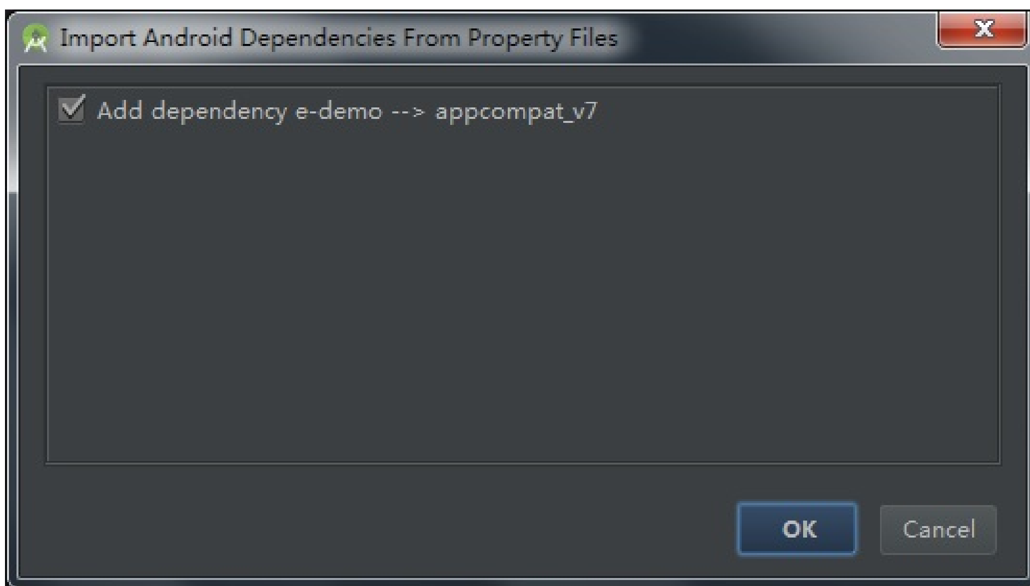


选择识别出的 **AndroidManifest.xml** 文件。默认全部勾选即可。

点击 **Finish** ，Android Studio就开始导入项目了。

讲解**18**

在导入的过程中，如果Android Studio识别出原项目的依赖关系，便会弹出对话框让你进行选择。如果想保持之前的依赖关系，点击 **OK** 即可。



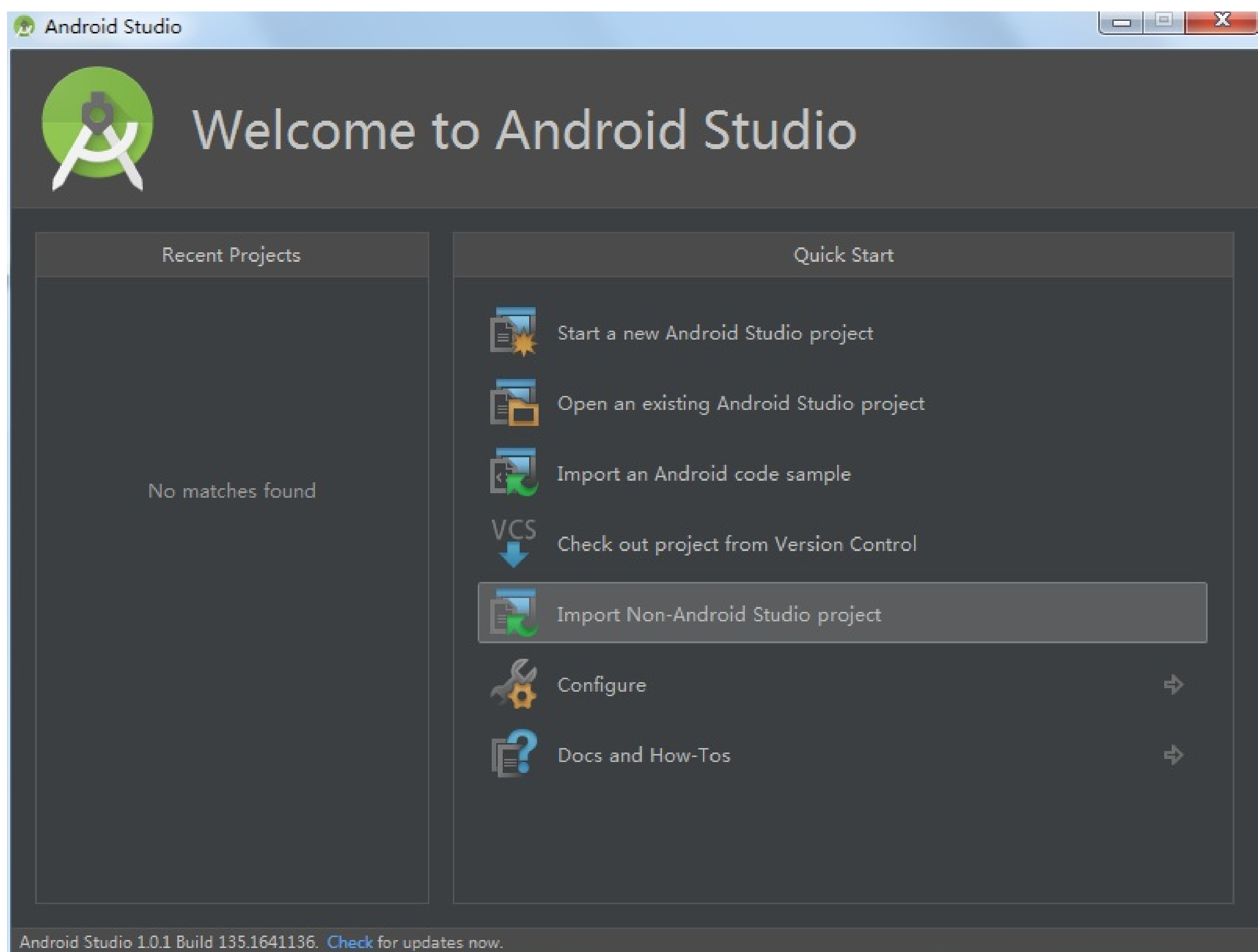
当你看到这个界面，就表示导入成功了。项目使用 **Ant** 构建，并不是Android Studio 默认的 **Gradle** 。

使用 Gradle 编译项目

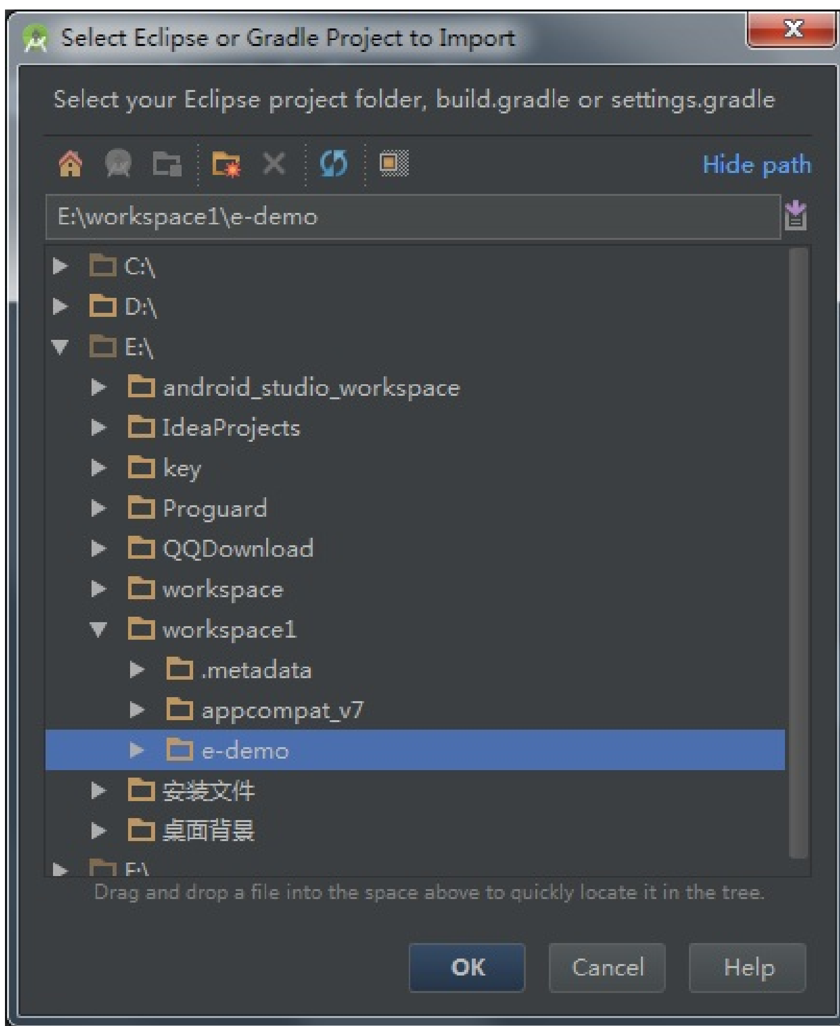
本方法有一个缺点就是，一次只能导入一个 **Eclipse** 项目。对于那些只使用到了官方系列的支持包的 **Eclipse** 项目来说，会方面很多，而且同时兼容 **Eclipse** 文件目录结构。

讲解**19**

打开Android Studio，选择 **Import Non-Android Studio project**。



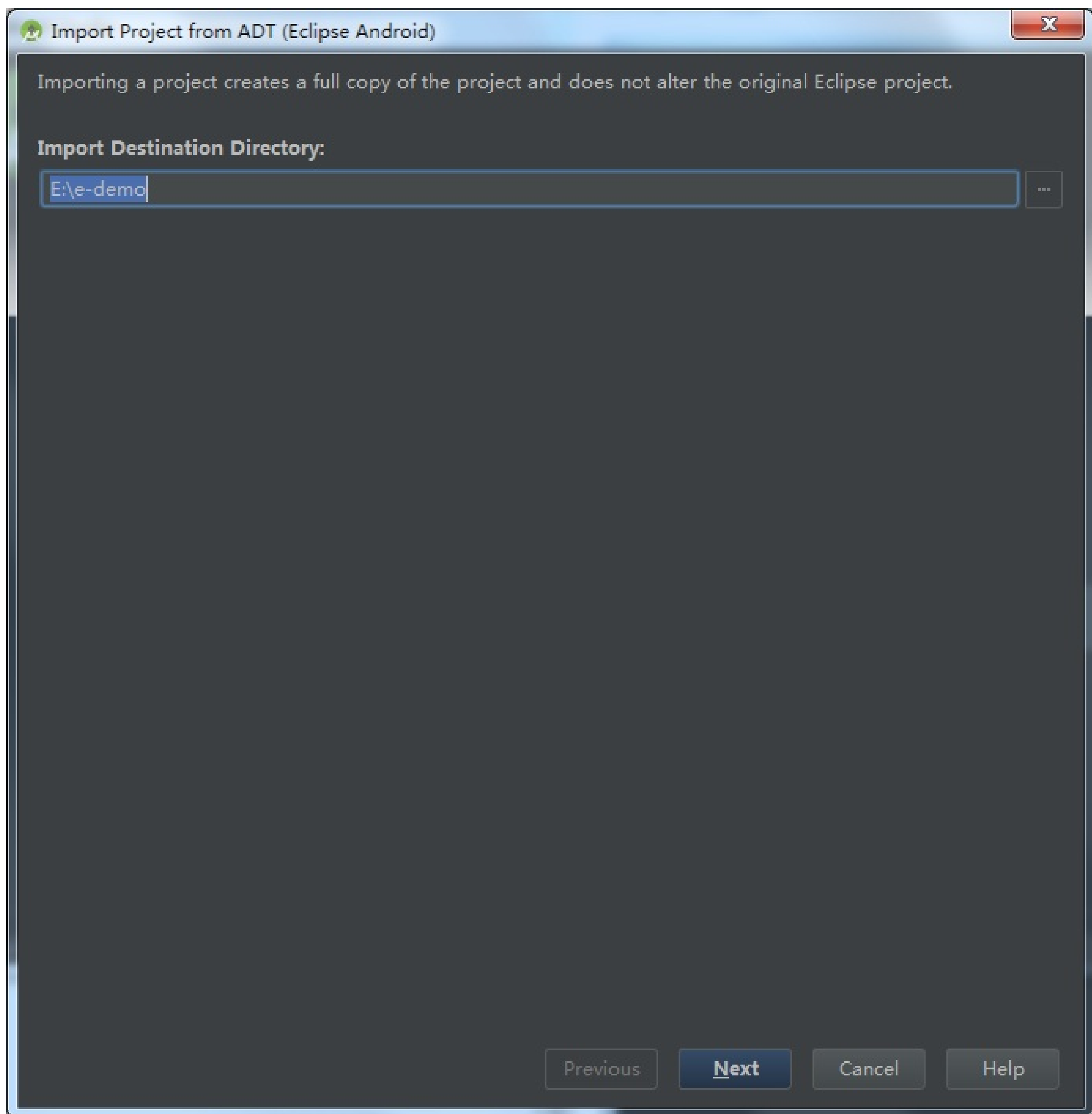
在弹出的目录选择框中，选择 你想导入的项目所在的目录，而不是 **Eclipse** 的工作空间 。



讲解**20**

Android Studio识别出你的项目是一个 **Eclipse Android** 项目，它将重新使用 **Gradle** 构建项目。

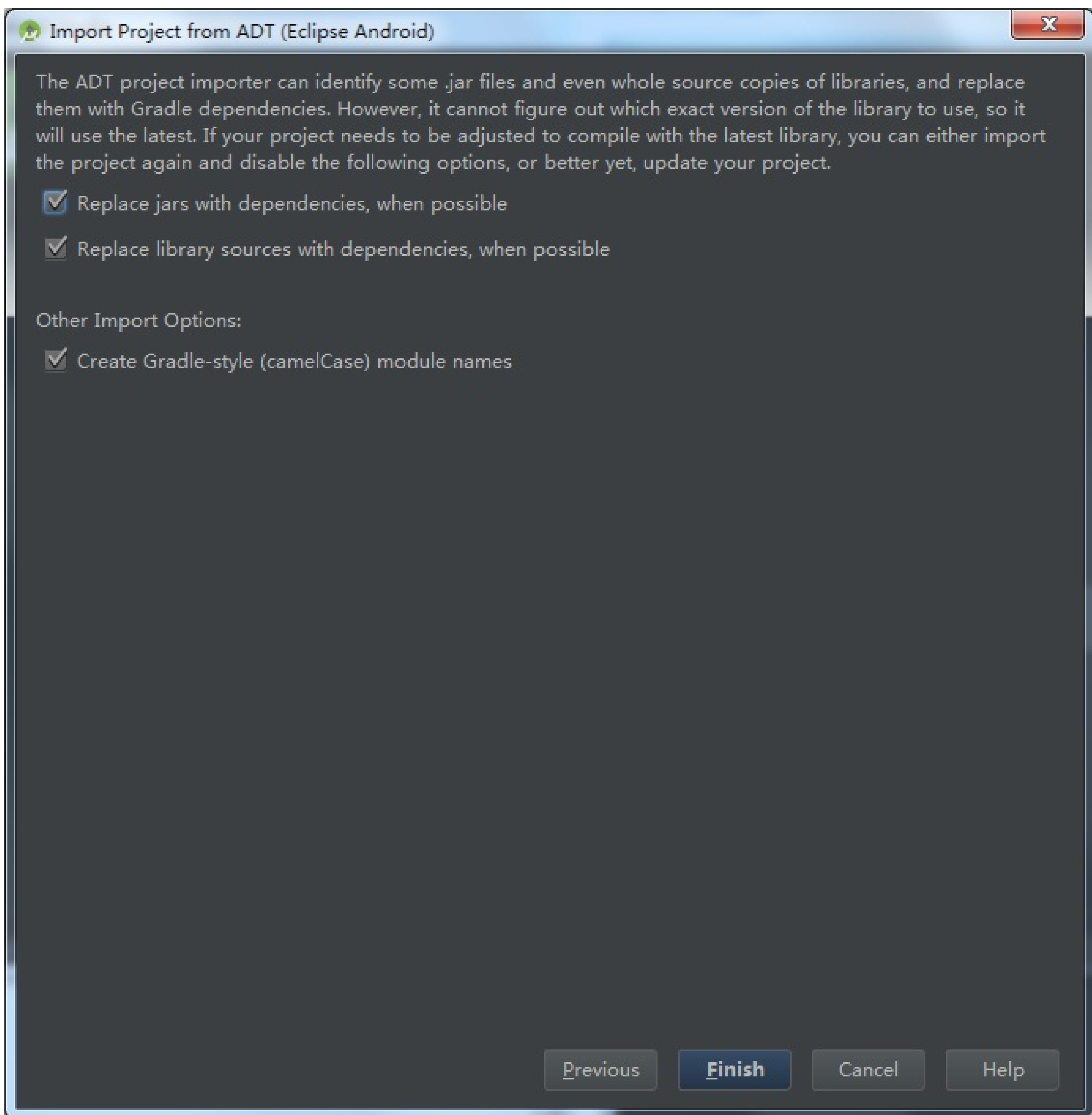
Android Studio会完整复制项目文件到一个新的目录中，你需要设置这个新目录的地址。



设置完毕，点击 **Next** 。

讲解**21**

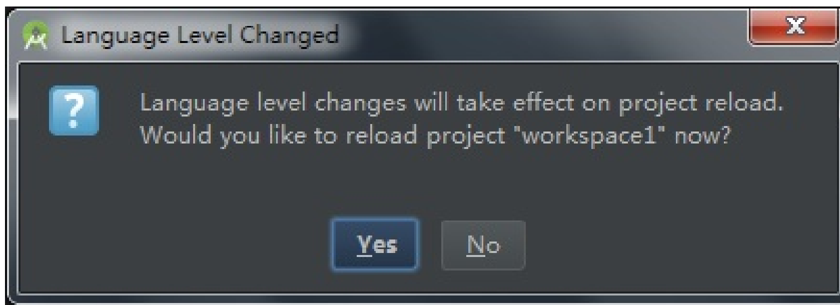
将之前 **Ant** 架构改变为 **Gradle** 架构。



- **Replace jars with dependencies,when possible** ：将 **Ant** 的 **jar** 依赖关系使用 **dependencies** 重构。
- **Replace library sources with dependencies,when possible** ：将 **Ant** 的 **library** 项目的依赖关系使用**dependencies** 重构。
- **Create Gradle-style(cameCase) module names** :使用 **Gradle** 的样式构建Android Studio的 **Module**名称。

建议全选，点击 **Finish** 。Android Studio开始编译项目，同时进入Android Studio主页面。

同样的，在编译的过程中，依然会提示你语言等级的问题，选择 **Yes** 。



导入Android Studio项目

Android Studio项目，指的是使用 **Gradle** 进行构建的项目。默认的文件结构如下：

```
project-name+ app/
 + src/
 + main/
 + assets/
 + java/
 + package
 ...
 + res/
 + drawable/
 ...
 + layout
 ...
 + values
 ...
 | Androidmanifest.xml
 | build.gradle
+ gradle/
 + wrapper/
 | gradle-wrapper.jar
 | gradle-wrapper.properties
| build.gradle
| gradlew
| gradlew.bat
| settings.gradle
```

有的一些Android Studio项目有对 **Eclipse** 做兼容处理，项目结构看起来和 **Eclipse** 是相同的。

导入Jar\/AAR

`.aar` 格式的包是Android独有的第三方库(Android Library), 包含了可重用的java文件和Android组件.我们可以通过新建module创建自己的Android Library,然后打包成.aar格式的包与别人共享,使用方法跟jar包基本一样.

导入步骤：

菜单栏: File —> New —> New Module… —> Import .JAR\/.AAR Package

选择aar包存放路径,并给予项目命名.

完成后可以看到作为模块导入的.aar文件.

mylibrary-debug的构建文件build.gradle:

```
configurations.create\("default"\)

artifacts.add\("default", file\('mylibrary-debug.aar'\)\)
```

Project的settings.gradle文件中自动添加了mylibrary-debug模块:

```
include ':app', ':phonetablemodule', ':mylibrary', ':mylibrary-debug'
```

新建类和文件

Android Studio新建类时可以先择多个常用的类模板.

操作步骤:

菜单栏 —> File —> New —> Java Class —> 输入类名,选择类模板

增加so文件

JNI是什么?

Android系统的底层库是由c\/c++编写，上层Android应用程序和应用程序框架通过JNI（JavaNative Interface)调用底层接口.

Android使用JNI开发分两种情况：一是使用已经编译好的.so动态库；二是使用c\/c++源代码开发.

一些第三方的库出于性能或代码安全的目的,会将核心代码用C\/C++来实现,然后提供编译好的so文件或jar包给我们.

so文件是什么?

Android中用到的so文件是一个c++的函数库，apk或jar包中调用so文件时，需要将对应so文件打包进apk或jar包.

如何加载**so**文件？

假设你的so文件为:libBaiduMapSDK_v350_1.so

这样加载:

String libName = "BaiduMapSDK_v350_1"; \/\/ 注意:库名libName, 没有前缀lib和后缀.so

System.loadLibrary(libName); \/\/在使用前一定要先加载

so文件应该放在哪里？

在Eclipse中我们将so文件放到libs目录下就可以了, 那么在AndroidStudio中应该将so文件放到哪里呢？

放到moduleName\/src\/main\/jniLibs目录下，如果没有jinLibs目录就新建一个.

armeabi、armeabi-v7a、mips、x86这四个文件夹是干嘛的呢?

表示四种不同的cpu类型，不同的cpu的特性不一样,在使用so文件时注意区分.

可以直接复制so文件粘贴到对应的文件夹下

ok之后so文件就进来啦.

创建Fragment文件

创建一个Fragment的一般步骤:

第1步: 在Layout目录下创建Fragment的布局文件.

第2步: 创建一个类文件,继承Fragment或者Fragment子类,并在onCreateView方法中加载布局文件.

第3步: 在Activity的布局文件中声明FragmentLayout.

使用模板基本上可以自动生成第1步和第2步.

如何使用Android Studio提供的模板快速创建Fragment及其所用到的布局文件呢？

操作步骤:

菜单栏: File —> New —> Fragment

Fragment(Blank):创建一个空白的Fragment

Fragment(List):创建一个空的Fragment,它包含一个网格列表.

Fragment(with a + 1 button):创建一个带有Google Plus +1按钮的Fragment

创建一个空白的Fragment

点击 【Fragment(Blank)】—> 然后弹出【New Android Component】界面;

- FragmentName: 自定义的Fragment类名, 会继承Fragment. 本例中为BlankFragment.
- Create Layout XML?: 如果勾选,会同时创建BlankFragment类对应的布局文件,并在BlankFragment类中自动添加加载该布局文件的代码.
- Fragment Layout Name: BlankFragment类对应的布局文件名,会根据类名自动生成,可自定义.
- Include fragment factory methods?: 如果勾选,会在BlankFragment类中生成工厂方法.
- Include interface callback?: 如果勾选,会在BlankFragment类中生成回调接口.

创建一个Fragment(List)

点击 【Fragment(List)】—> 然后弹出【New Android Component】界面:

- Package name: 包名
- Object Kind: 对象类型
- Fragment class name: Fragment类的名字
- Column Count: 网格的列数
- Object content layout file name: 对象内容布局文件名
- List layout file name: 列表布局文件名
- Adapter class name: Adapter类名

创建service文件

操作步骤:

菜单栏: File —> New —> Service

然后弹出【New Android Component】界面:

在这里可以创建一个service组件,并添加到AndroidManifest.xml文件中.

- Class Name: 类名,会继承Service.
- Exported: Service的属性,表示是否支持其它应用调用当前组件.
- Enable:Service的属性,表示该服务是否能够被实例化.

使用默认配置,然后创建成功.

创建自定义组件

有时Android提供的组件无法满足我们的需求,因此需要自定义组件.

创建自定义组件的一般步骤:

1.新建类文件,要继承View或View的子类.

2.覆写父类的一些方法.

3.使用自定义组件类.

Android Studio会通过模板帮我们自动生成文件,并覆写方法. 我们只需要根据自己的实际需求修修改改就好了.

操作步骤:

菜单栏: File —> New —>UI Component —> Custom View

然后弹出【New Android Component】界面:

- Package name: 包名
- View Class: 自定义的类名,按约定应该以View结尾.

使用默认配置,点击【Finish】后创建成功.

模板帮我们继承了View并覆写了方法,然后我们根据自己的需求修改代码就可以了.

另外在layout目录下会自动生成一个sample_my_view.xml文件.这是一个示例,用来告诉你自定义组件怎么用.

创建app widget

App Widget是应用程序的窗口小部件,它可以被嵌入到其它应用程序中(如桌面)并接收周期性的更新.

创建Widget的一般步骤:

第1步:在res\/layout目录下创建一个Widget布局文件.

第2步:创建一个类继承AppWidgetProvider.

第3步:在res\/xml目录下创建一个XML文件,用来定义Widget的特性.

第4步:在AndroidManifest.xml中声明Widget.

使用Android Studio的模板功能,可以帮我们自动完成上面这些步骤.

操作步骤:

菜单栏: File —> New —> Widget —> App Widget

然后弹出【New Android Component】界面:

Class Name: 类名,继承AppWidgetProvider.[图片]Placement: Widget 放在哪儿.

1.Home-screen and Keyguard: 在主屏幕和锁键上.

2.Home-screen only: 仅在主屏幕上.

3.Keyboard only(API 17+): 仅在锁键上(只支持Android4.2及以上版本).

Resizable(API 12+): Widget是否可调整大小,只支持Android 3.1及以上版本.

1.Horizontally and vertically: 水平和垂直显示时可调整.

2.Only Horizontally: 仅水平时可调整.

3.Only vertically: 仅垂直时可调整.

4.Not resizable: 不可调整.

- Minimum Width: 最小宽度,参照左边预览窗口的单元格.
- Minimum Height: 最小高度,参照左边预览窗口的单元格.
- Configuration Screen:勾选后会生成widgets配置activity.

使用默认配置,点击【Finish】后创建成功.

创建可编译的资源文件

Android项目中可编译的资源文件存放在res目录下,在R.java中会自动生成这些资源文件的ID,可以通过R.XXX.ID来访问。

光标放在不同的文件夹上,新建列表中显示的选项是不同的。

1.如果光标放到**module**上**.**

操作步骤**:**

菜单栏: File | 右击module —> New —> Android resource file

弹出[New Resource File]对话框

File name:

新建的资源文件名.

Resource type:

资源类型,下拉列表中会列中我们前面讲过的缺省的资源文件类型.

不同的资源类型会有不同的根元素和目录名,根元素和目录名会根据你选择的文件类型而变化.

XML文件的根元素,不同的资源类型有不同的根元素,在新建文件以后会自动生成.

例1:资源类型为Values的文件,根元素为resources

例2:资源类型为Transition的文件,根元素为transitionManager

Source set:

编译时引用的资源文件的来源设置,缺省的是main、debug、release.

资源文件会放在不同的文件夹下面,在打不同的包时会引用不同的资源.

Directory name:

资源文件的目录名与资源类型一一对应,不可更改,新建的资源文件会自动放到这个目录下面,如果目录不存在会自动创建.

Available qualifiers:

可用的资源限定符,具体请看《2.17 资源限定符》

如果光标放到资源文件夹上:

会显示对应的资源文件新建选项.

点击后也会提示新建对应的资源文件,这样目标更明确.

3.新建资源文件目录**:**

新建资源文件目录只是新建目录,没有新建文件的地方.配置项里目录名可以自定义.

资源限定符

在新建资源文件时可以选择资源限定符.

在Available qualifiers列表中列出了适配时常用的一些限定符,选择限定符,再进行相应的配置,就可以在res目录下生成相应的资源文件夹.

Country Code:

移动设备国家代码:唯一识别移动用户所属的国家,共3位,中国为460;

移动设备网络代码:是与Country Code相结合，以用来表示唯一一个的移动设备的网络运营商.

Locale：

国际化(多语言):如果用户改变了系统中的语言设置，那么在应用程序的运行期间也能够改变为对应的语言.

Layout Direction：

布局方向.

LTR: 从左到右 RTL: 从右到左

Smallest Screen Width:

最小屏幕宽度:这个值是布局支持的最小宽度，而不管屏幕的当前方向.

当应用程序提供了多个带有不同值的最小宽度限定符资源目录时，系统会使用最接近(不超出)设备最小宽度的那个资源.

Screen Width:

最小的可用屏幕宽度，当方向在横向和纵向之间改变时，这个配置值会跟当前的实际的宽度相匹配.

Screen Height:

最小的可用屏幕高度，当方向在横向和纵向之间改变时，这个配置值会跟当前的实际的高度相匹配.

Size:

屏幕尺寸.

Small: 小屏幕尺寸,类似低分辨率的QVGA屏幕,大约是320x426dp;

Normal: 中等屏幕尺寸,类似中等分辨率的HVGA屏幕,大约是320x470dp;

Large: 大屏幕尺寸,类似中等分辨率的VGA屏幕,大约是480x640dp；

X-large: 超大屏幕尺寸,比HVGA屏幕大,大约是720x960dp;

Ratio:

宽高比率：指的是实际的物理尺寸宽高比率

long: 长屏幕 nolong: 非长屏幕

Orientation:

限制横竖屏切换.

portrait竖屏、landscape横屏、square正方形

UI Mode:

UI模式.

Car Dock:车座、Desk Dock:桌座、Television:电视上、Appliance:装置、Watch:手表

当用户将手机放在不同的Dock上或插入不同的地方,应用程序在运行期间就能够改变这个限定.

Night Mode:

夜间模式.

Not Ntight: 白天

Night: 夜间

Density:

密度：

指定不同密度的手机使用不同的资源,如果没有提供与当前设备配置匹配的可选资源，那么系统会使用最接近的资源.

Touch Screen:

触屏类型:

No Touch: 非触屏

Stylus: 触控笔

Finger: 手指(触屏设备)

Keyboard:

键盘类型.

Exposed: 设备有可用的键盘(硬或软)

Hidden: 设备有可用的硬键盘（被隐藏,且无可用的软键盘）

Soft: 设备有可用的软键盘(不管否可见)

Text Input:

文本输入法

No keys: 设备没有用于文本输入的硬键盘

Qwerty: 设备有标准的硬键盘,不管用户是否可见

12 Key: 设备有12个键的硬键盘,不管用户是否可见.

Navigation State:

导航键的状态.

Exposed: 导航键可用

Hidden: 导航键不可用

如果用户能够看到导航键，那么在应用程序运行时就能够改变这个限定.

Navigation Method:

导航方法:None:除了使用触屏以外,设备没有其他导航设施、 D-pad:设备有用于导航的定向板、 Trackball:设备有用于导航的轨迹球、 Wheel:设备有用于导航的定向滚轮

Dimension:

自定义屏幕尺寸

Version:

版本.

设备支持的API级别

创建AIDL

什么是AIDL？

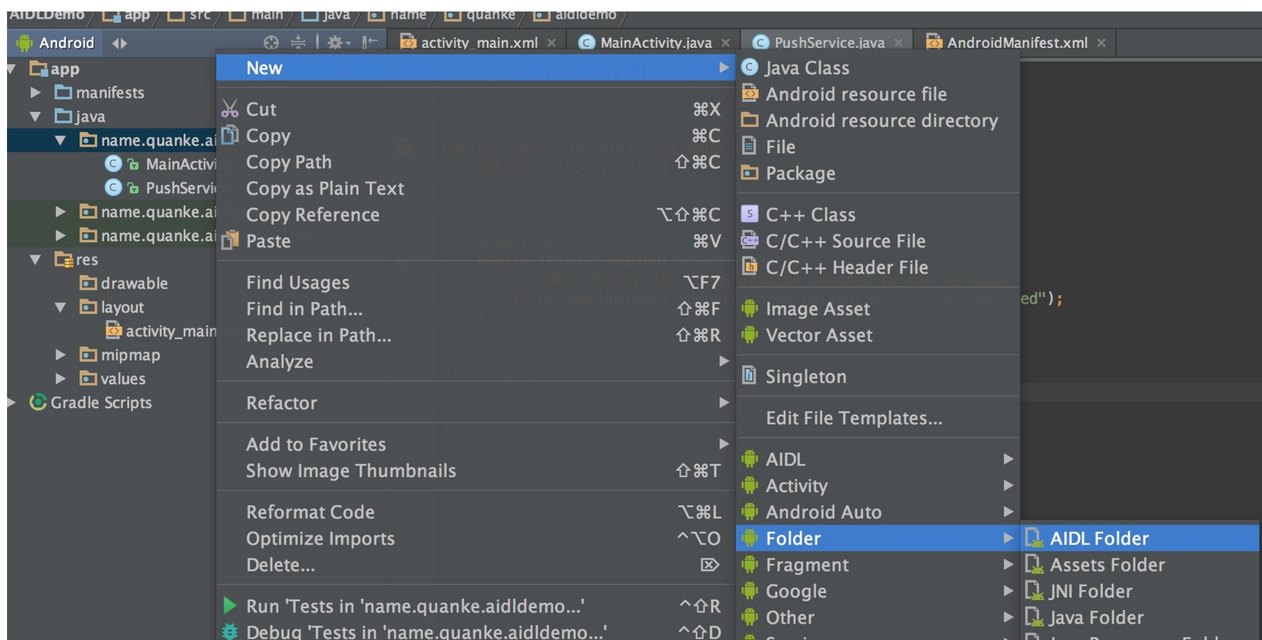
`AIDL` 是 `Android Interface definition language` 的缩写，它是一种 `Android` 内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口

AIDL可以解决什么问题？

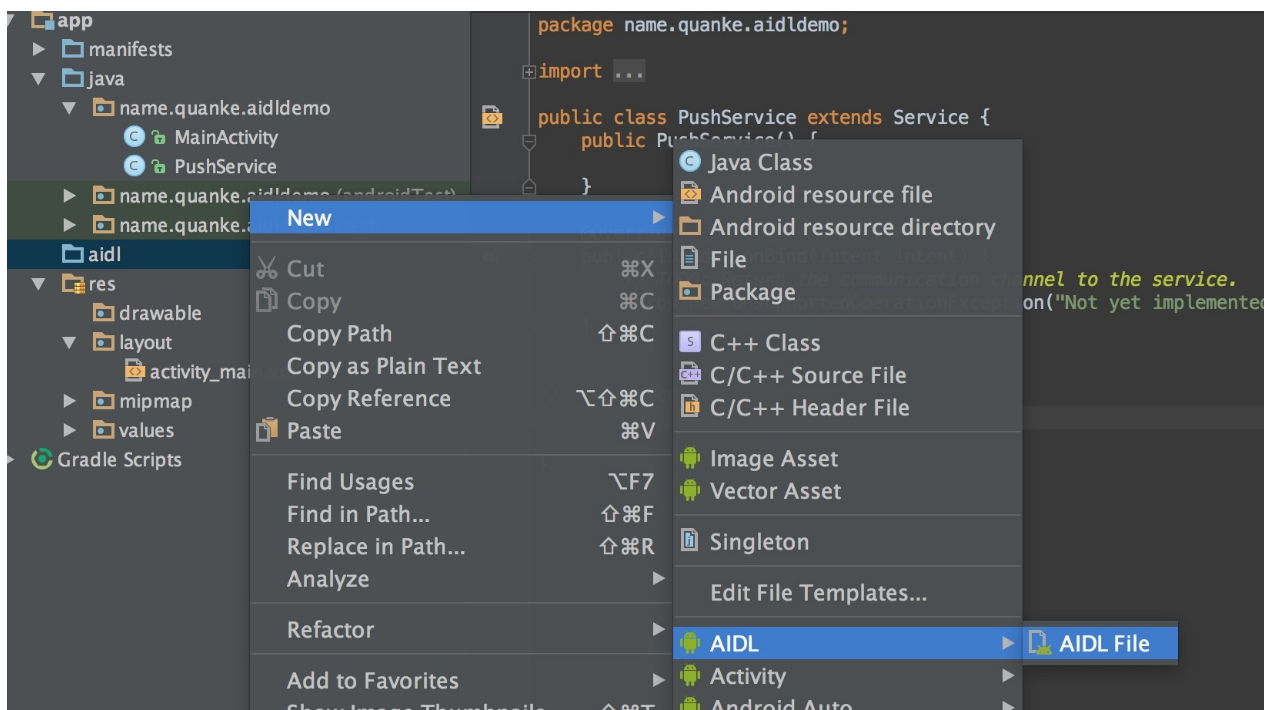
- 可以实现多个应用程序共享同一个Service的功能，比如：IM服务可以提供给多个APP使用，先在推送基本都是采取这种方案
- 可以跨进程调用服务里的方法

实战演示：

1.创建AIDL文件夹



2.创建AIDL文件



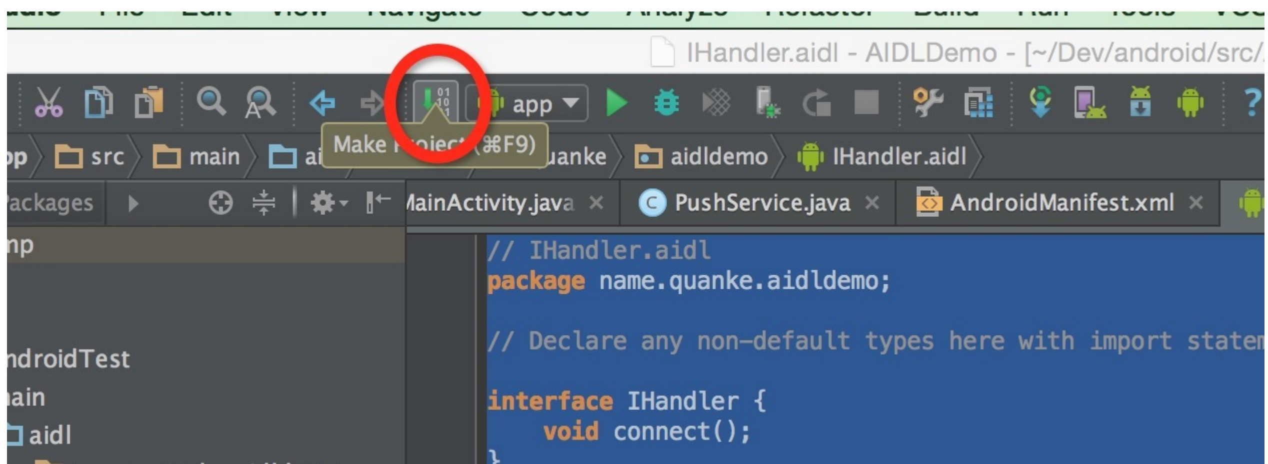
3.编写AIDL文件

```
// IHandler.aidl
package name.quanke.aidldemo;

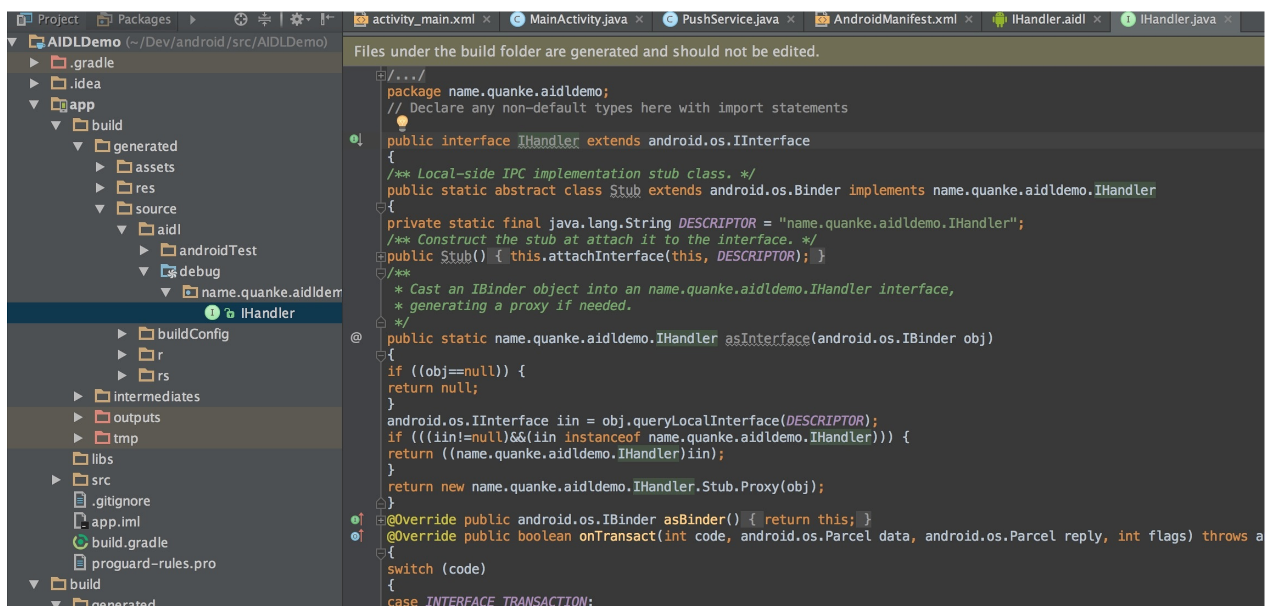
// Declare any non-default types here with import statements

interface IHandler {
    void connect();
}
```

4.AIDL文件 生成接口



生成后的样子

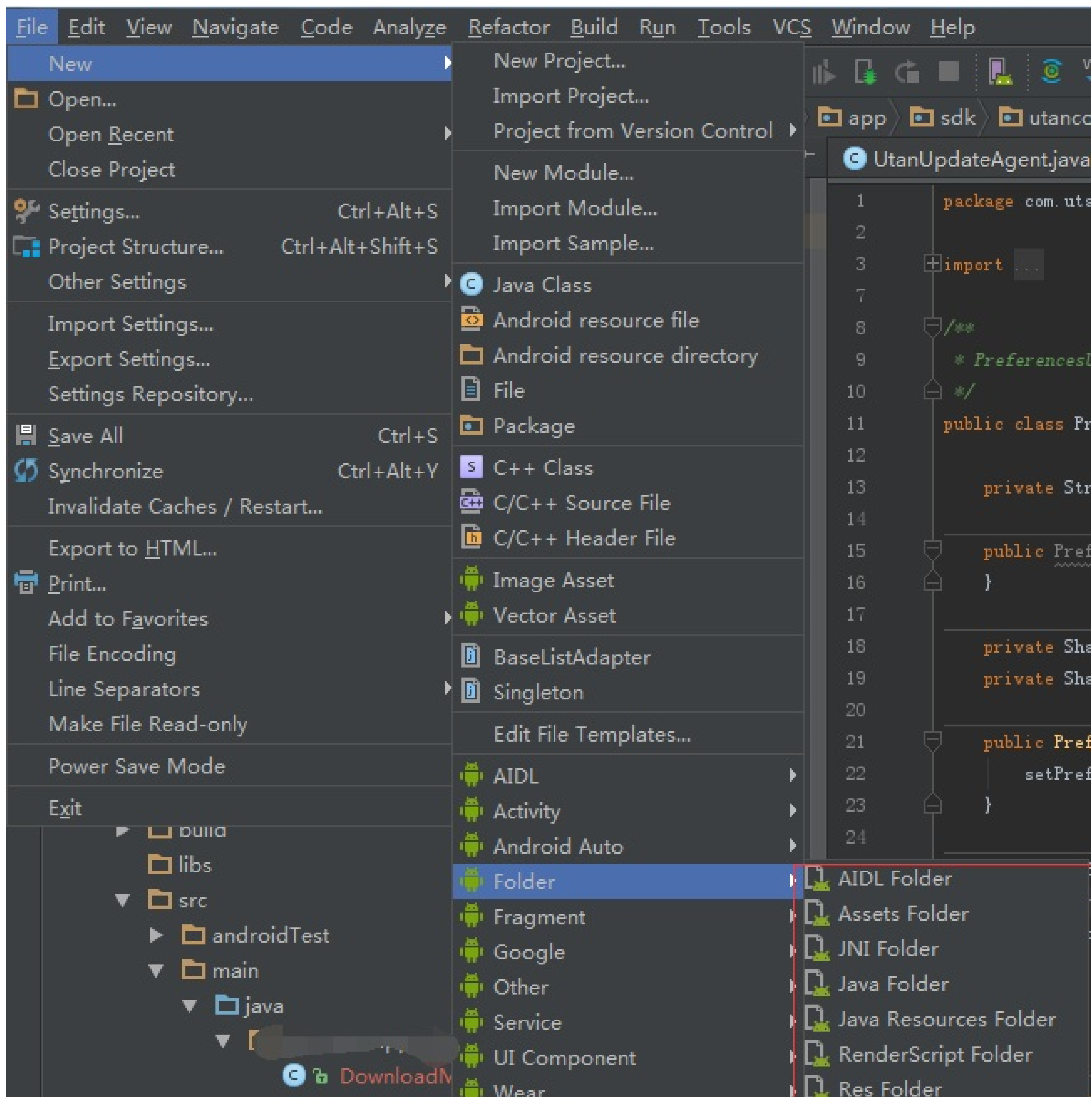


> 关于AIDL的使用，可以看我的博客，有详细讲解

http://quanke.name/2016/07/22/Android-Studio-Service-AIDL-%E8%AF%A6%E8%A7%A3/

创建Android文件夹

在Android Studio中提供了新建文件夹的功能,在这里可以新建一些缺省的文件夹.



第三章 布局

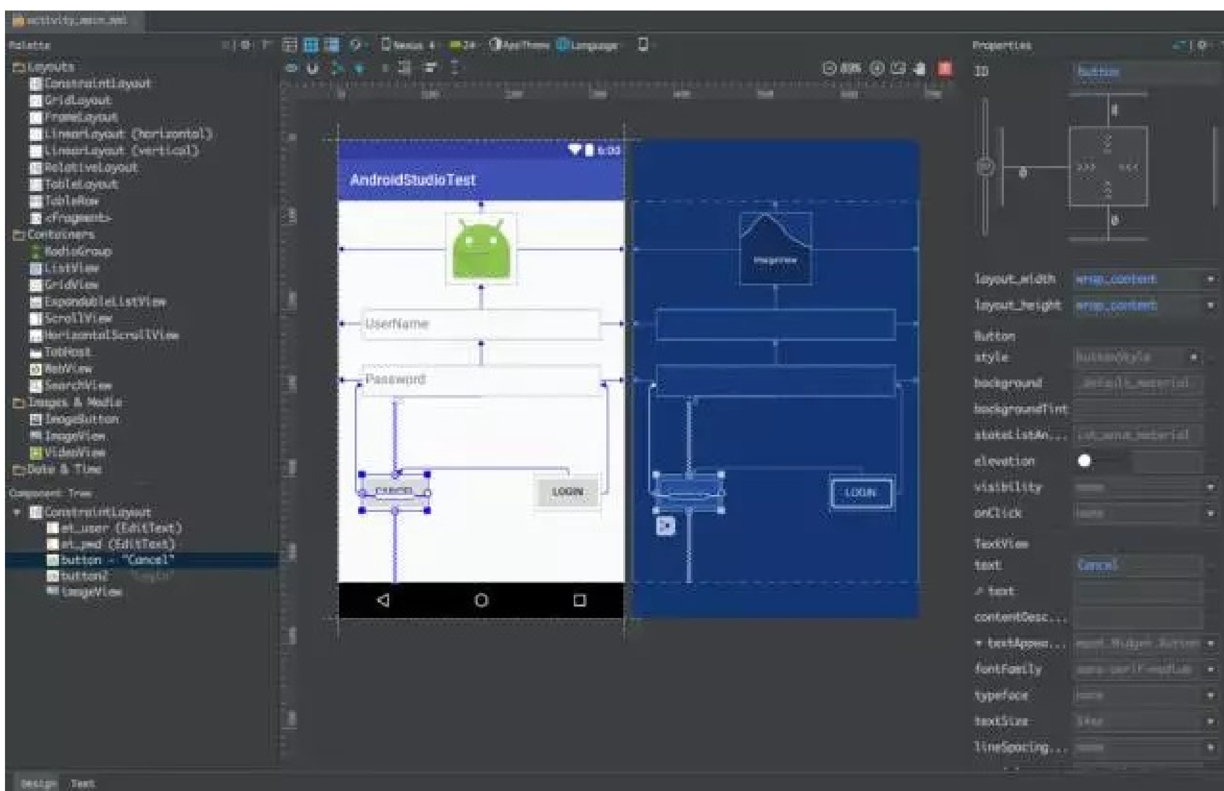
Android Studio中创建布局文件有两种方式: 在XML文本编辑器中直接编写代码和使用可视化布局编辑器通过拖拽控件自动生成。

可视化布局编辑器可以通过拖拽控件生成布局，可以实时预览布局的效果。可以在不安装APP的情况下，随时查看布局在不同分辨率、不同版本、不同品牌手机上的显示效果， 节省了大量调试界面的时间。

ConstraintLayout约束

1.ConstraintLayout基本界面

更新Android Studio 2.2之后，更新了布局设计器，同时，引人了ConstraintLayout，这一布局，旨在降低布局层级，其主要界面如下所示：



这个界面主要分成下面几个部分：

- 左侧边栏，包括Palette组件库和Component Tree
- 中间是布局设计器，包括两部分，左边是视图预览，右边是布局约束
- 右侧边栏，上面是类似盒子模型的边界和大小布局设计器，下面是属性列表

在熟悉了界面之后，我们要做的就是理解，什么是ConstraintLayout。ConstraintLayout的核心，实际上就是『约束』，这个翻译很直接，也很准确，它可以说是一个强化的RelativeLayout，只不过比RelativeLayout增加了更多的约束条件和方式，从这一点上去理解，就很容易接受了。

> 在第一次引人ConstraintLayout的时候，Android Studio会自动去下载依赖，等他自动完成安装即可。 最后，在build.gradle中会添加一行依赖：
>
> ```
> compile 'com.android.support.constraint:constraint-layout:1.0.0-alpha8'
> ```

Google提供了一个CodeLab来帮助开发者熟悉这个布局，地址如下所示：

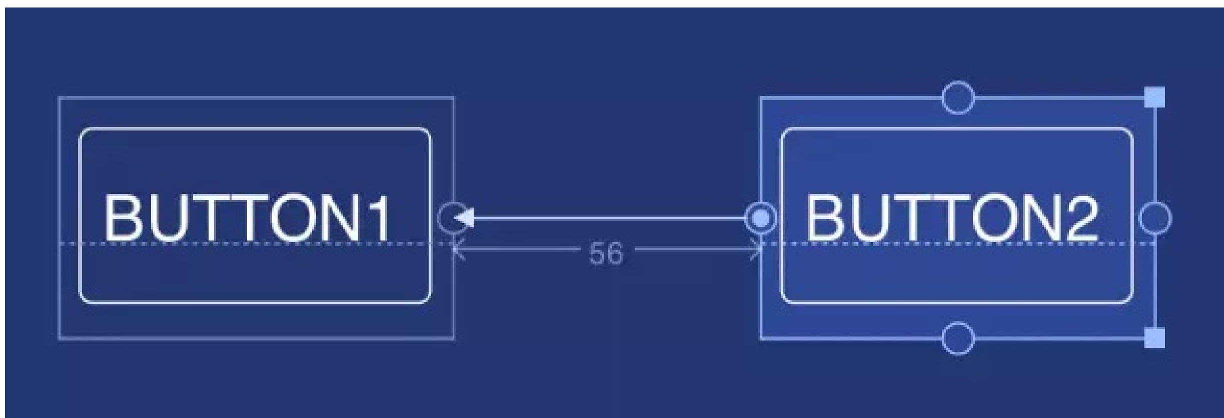
https:\/\/codelabs.developers.google.com\/codelabs\/constraint-layout\/index.html#0

同时，2016IO上Google也给出了一个Topic来讲解，地址如下所示：

https:\/\/youtu.be\/sO9aX87hq9c

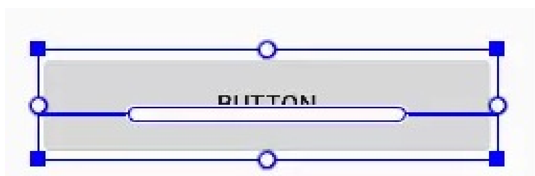
2.ConstraintLayout约束类型

简单的说，约束，就是组件与组件之间的关系，借用官网上的一张图，我们来解释下：



这里展示的，就是左右两个Button直接的关系，这实际上就是一个简单的相对布局方式，下面我们来看一下具体的约束类型。

当我们点击一个控件的时候，它的显示效果如图所示：



这里主要包含几种类型的约束

- 尺寸约束
- 边界约束
- 基准线约束

我们一一来看。

尺寸约束

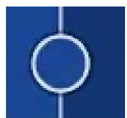
尺寸约束使用的是『实心方块』，如图：



这个很好理解，就是调整组件的大小。

边界约束

边界约束使用的是『空心圆圈』，如图：



边界约束，是使用最多的约束，它用于建立组件与组件之间、组件与Parent边界之间的约束关系，实际上，就是确定彼此的相对位置。

基准线约束

基准线约束，使用的是『空心圆角矩形』，如图：



基准线约束，是让两个带有文本属性的组件进行对齐的，可以让两个组件的文本按照基准线进行对齐。唯一要注意的是，你需要把鼠标放在控件上，等基准线约束的图形亮了，才可以进行拖动。

清除约束

通过工具栏上的『清除约束』按钮，或者是控件上的悬浮提示，都可以清除一个控件的所有约束条件，如图：

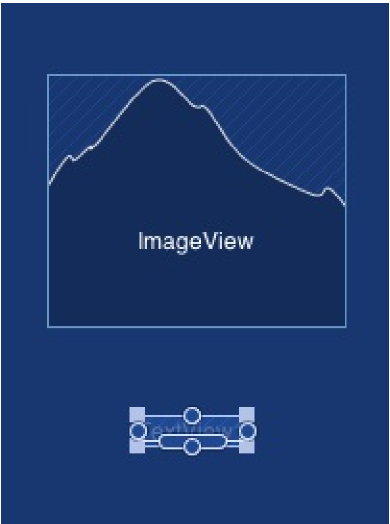


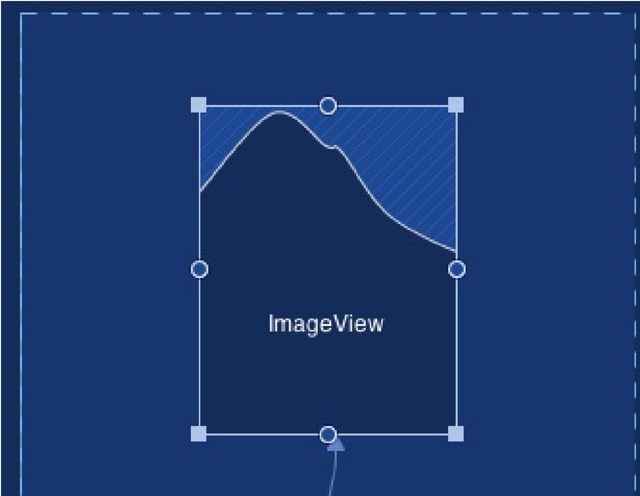
掌握好这几种约束条件的使用后，就可以自己去尝试下了，我们只要拖一个控件，来体验下。

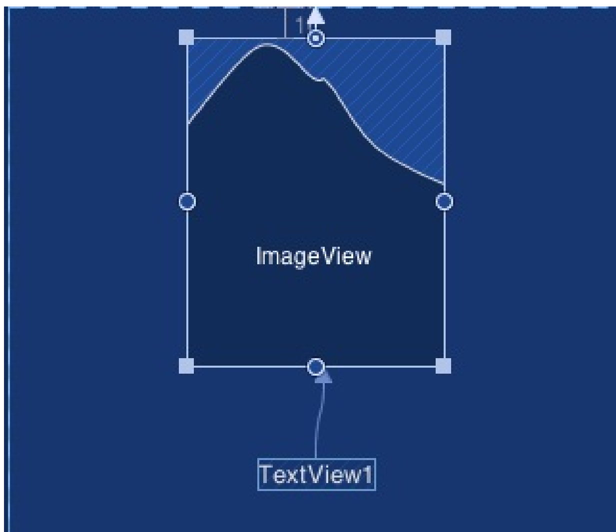
3.约束示例

这里我把官网上的几个Demo的动图Copy过来：


TextV


ImageView

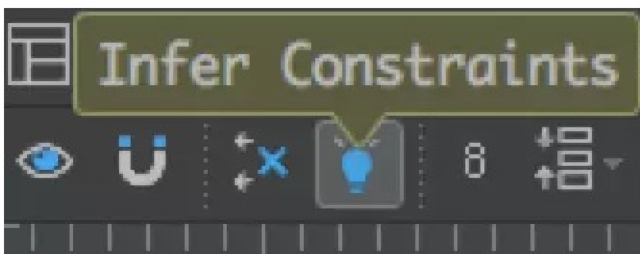

ImageView



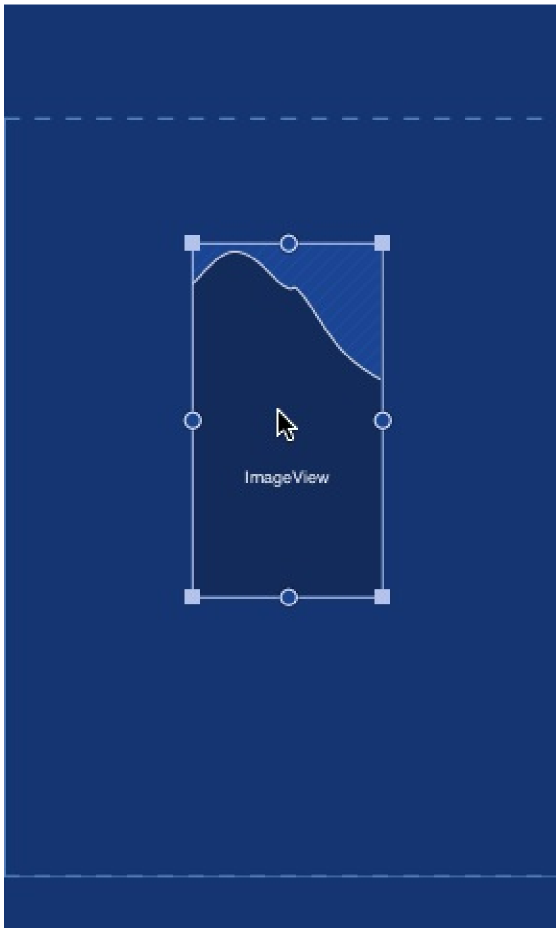


4.自动约束Autoconnect

在布局设计器的菜单栏上，有一个『磁铁』一样的图标，如图：



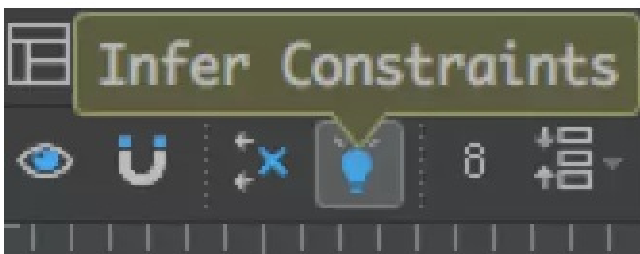
默认这个按钮就是打开的，通过这个，我们可以实现组件约束的自动创建，Demo示例如图：



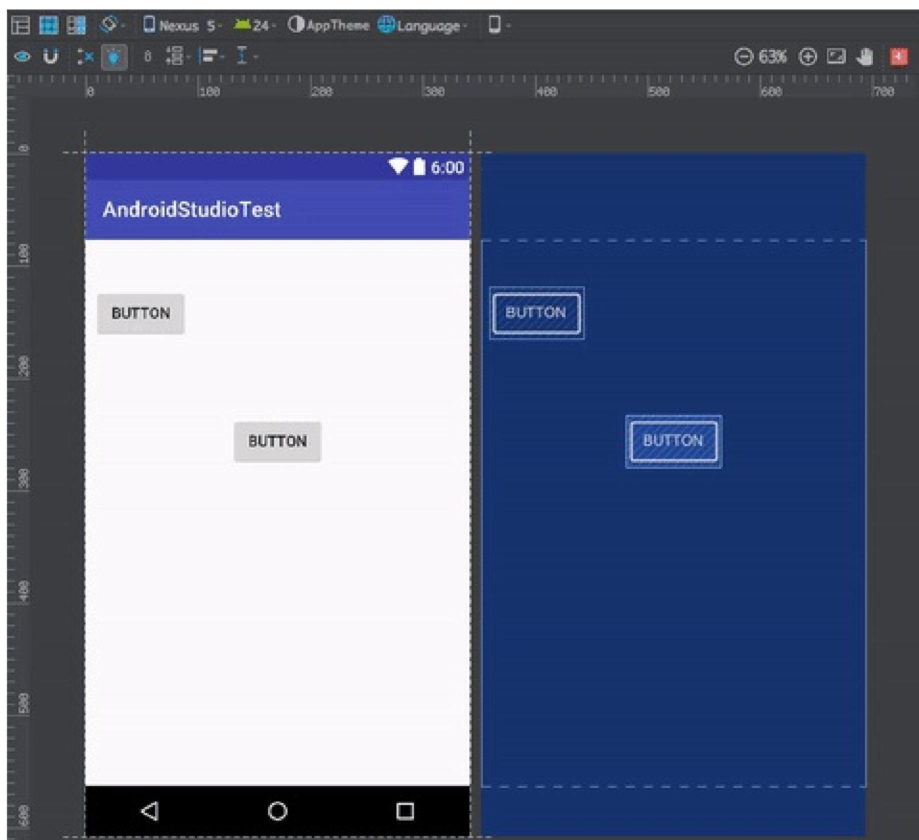
这个和PPT里面拖动布局的时候，会弹出对齐的基准线，然后帮你自动居中这些功能类似。实际测试下来，这个功能可以很方便的在拖动组件的时候，帮你写好约束，但有些精确的调整，还是需要手动去创建的。

5.约束推断Inference

在布局设计器的菜单上，还有一个『灯泡』一样的按钮，通过这个按钮，可以帮我们自动创建组件间的约束关系，他分析的是一个组件附近的组件，并根据当前在设计面板中的位置来创建约束关系，如图所示：

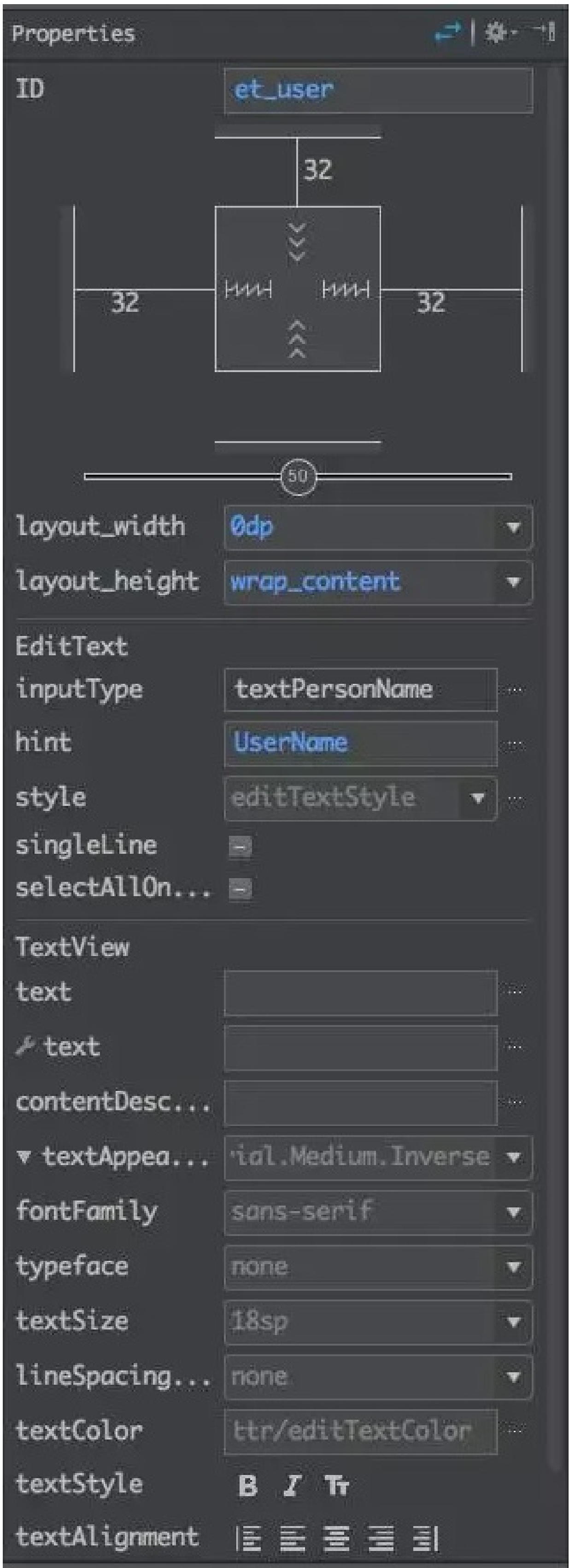


约束推断这个功能非常强大，我们只需要把组件拖到一个地方，然后就可以通过推断，来完成最基本的约束创建，最后，手动进行完善即可。

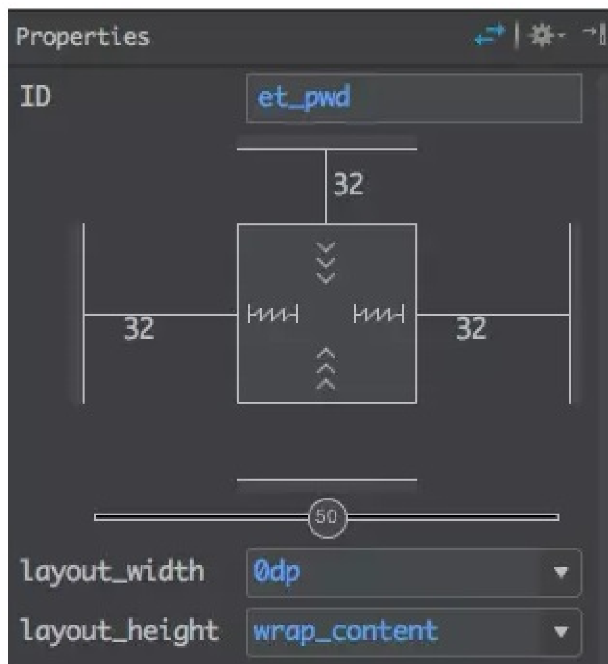


6.View Inspector

Inspector界面就是设计布局的右边栏，包含了一个类似盒子模型的布局检查器和对应属性的属性列表，如图所示：



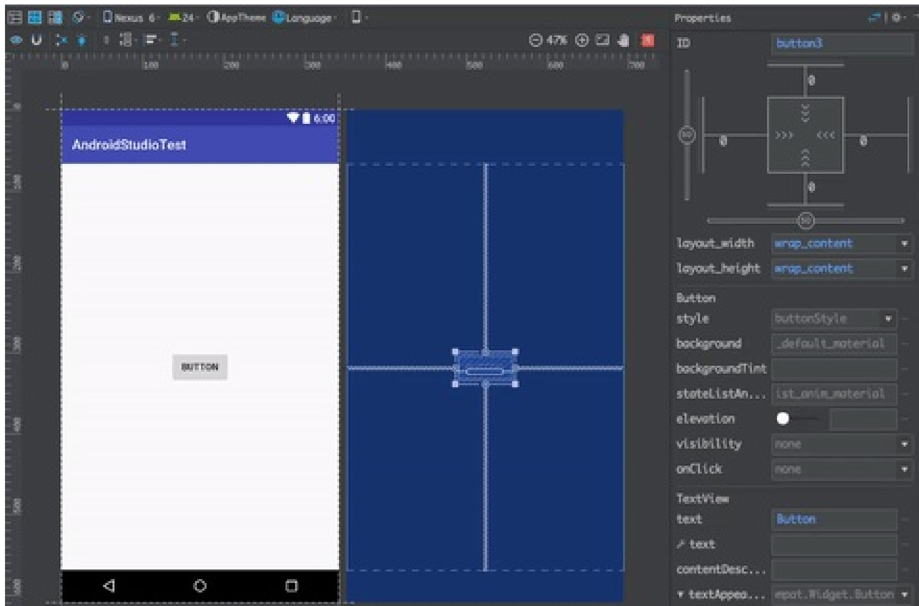
属性这一块我们就不看了，和大家在XML中写的属性是一样的，只不过这里通过可视化的方式弄出来了，这个之前就有了，我们主要来看下上面的那个界面。



这上面的ID，不多说了，这个盒子四周的线，代表着我们的Margin设置，在工具栏上，还可以设置Margin的基数，对于MD设计风格，这个基数一般是8dp，所以，这里可以选择X8的Margin：

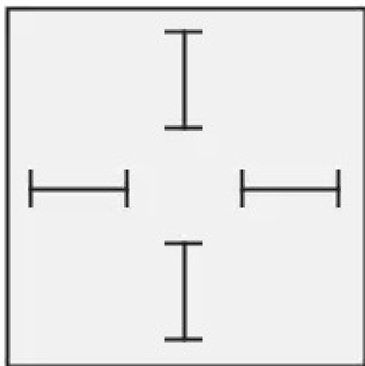


另外，最外面边框上还有两个带数字的小圆圈，这个就是控制相对位置的比例的，如图：



通过这个比例的设置，我们天然就自带了百分比布局。

最后，最难理解的就是盒子里面的那四根线，如图：



这里的四根线，在点击后，会发生变化，总共有以下几种：

Fixed



这样一个类型的线，可以让你写定具体的大小数值。

Wrap Content



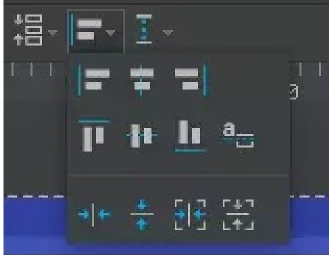
这个就是Wrap Content的含义，包裹内容，没有发生变化。

AnySize



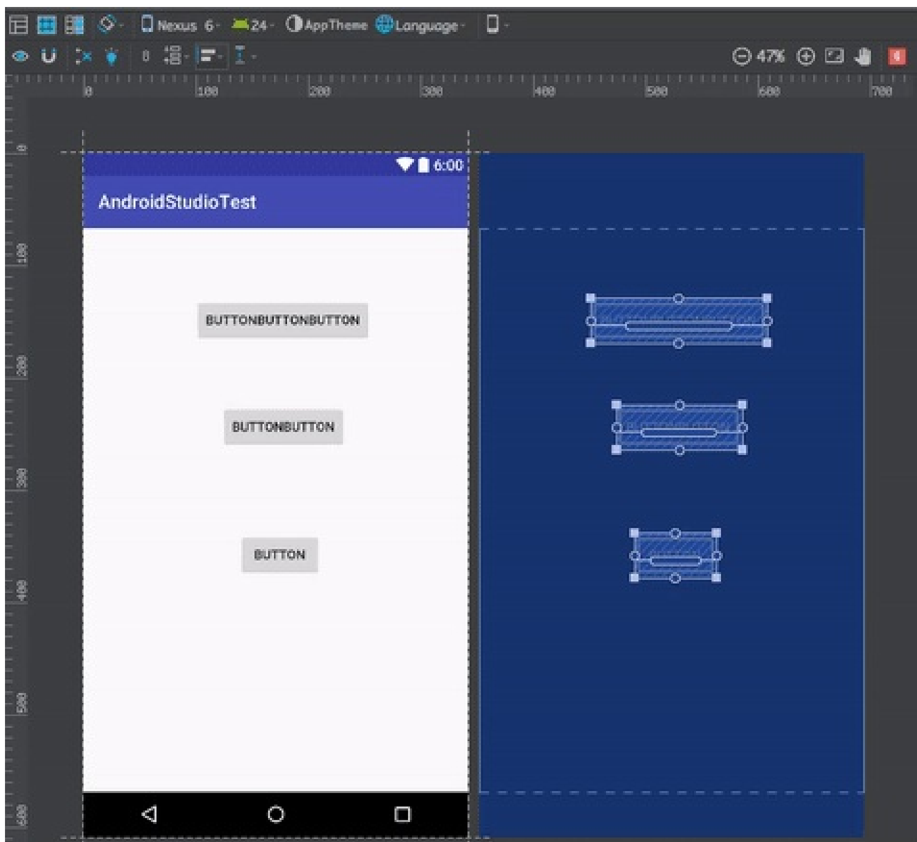
这个就是最难理解的，它表示组件会占用所有的可用空间来适应约束，类似线性布局中，设置width=0，weight=1的方式。

7.Align



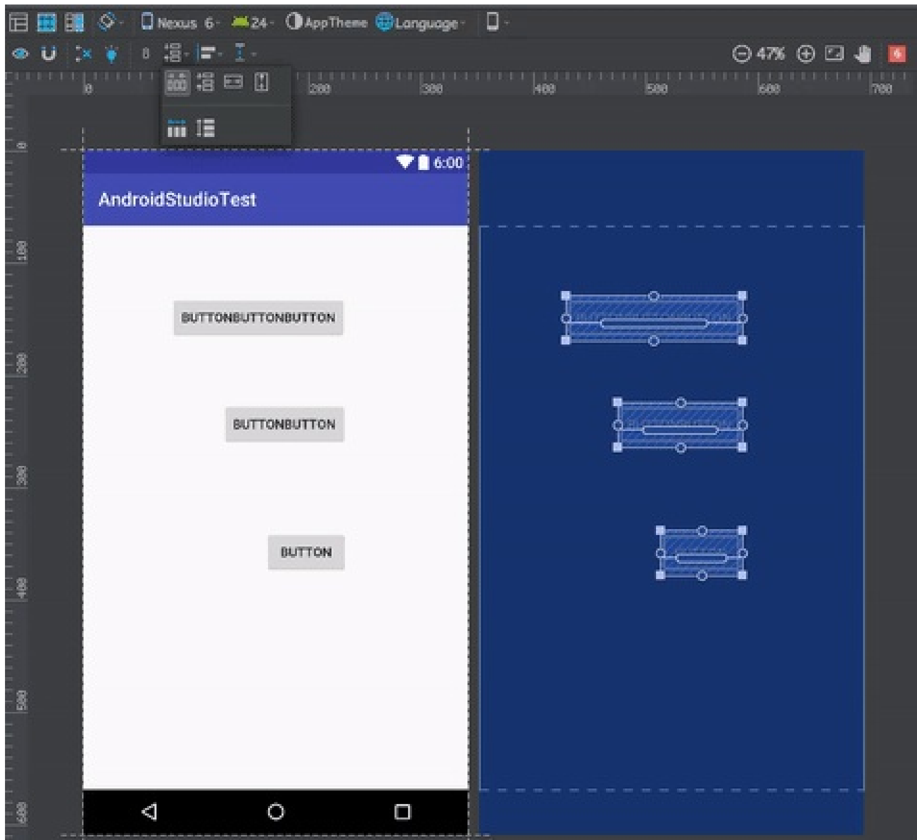
在工具栏中，可以使用对齐工具，快速给选定组件设置对齐约束，如图：

我们可以来演示下：



8.Pack

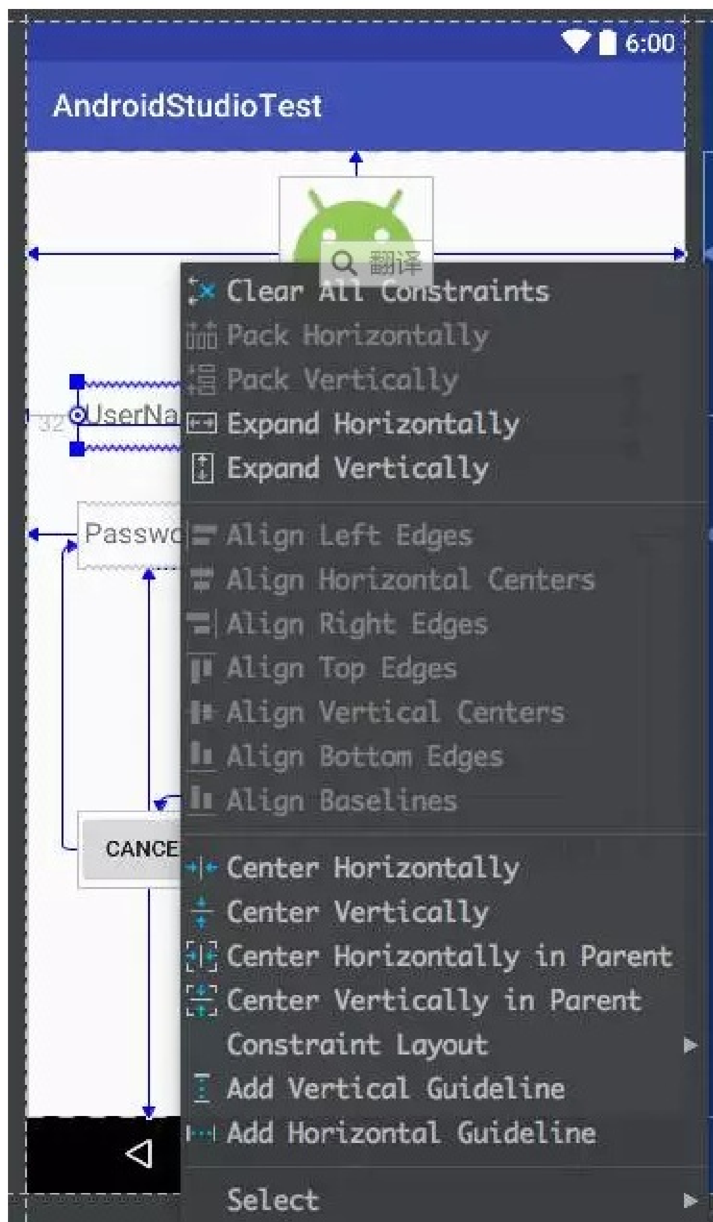
在工具栏中，可以使用Pack工具，快速对组件进行编组操作，如图：



9.快捷布局

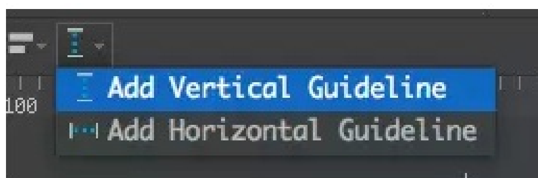
在一个组件上点击右键，可以快速创建一些布局的快捷设计，如图所示：

在这里，可以快速设置组件的居中，对齐等方式。

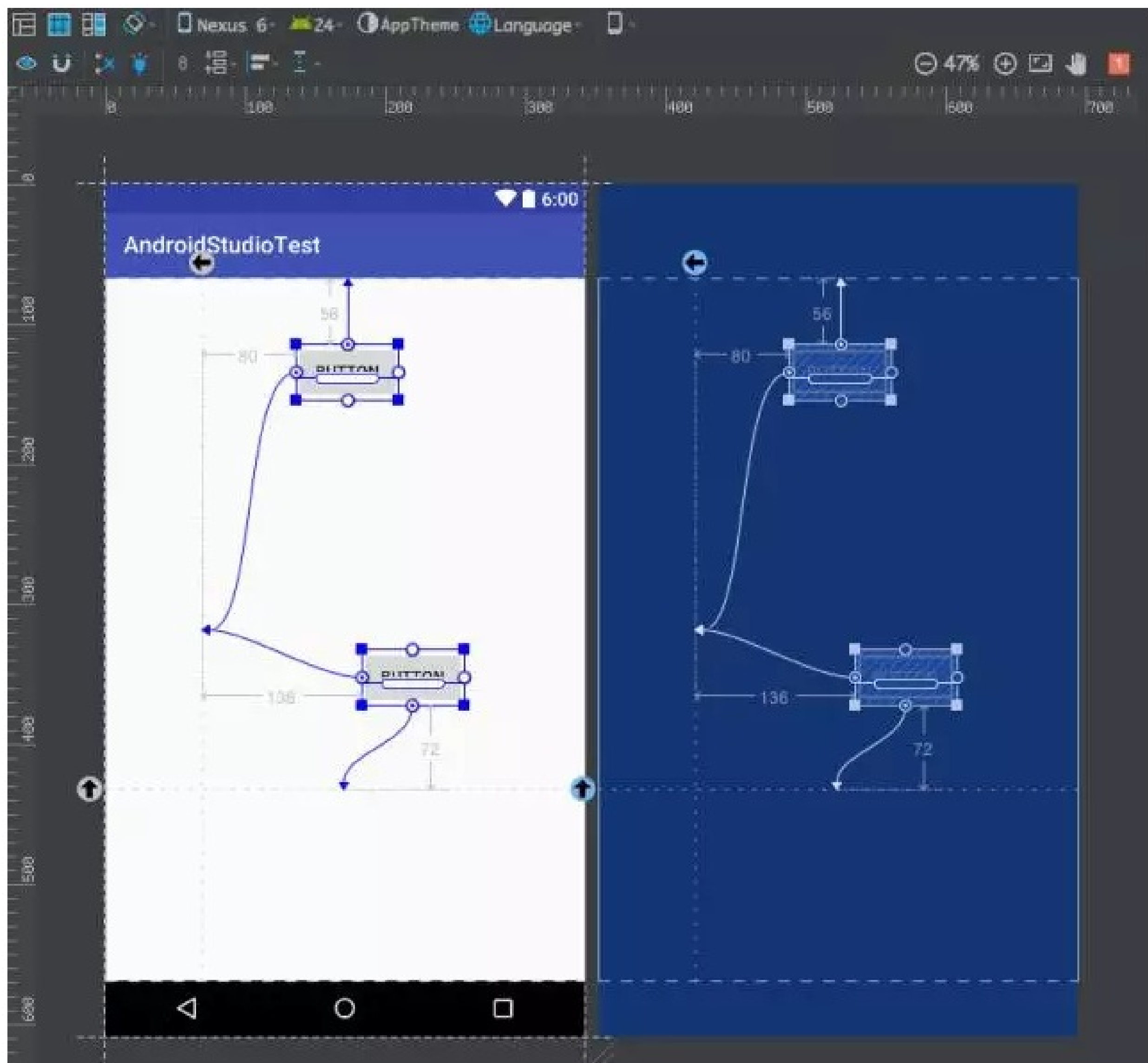


10.GuideLine

为了更加灵活的布局，ConstraintLayout还提供了一个GuideLine，如图所示：

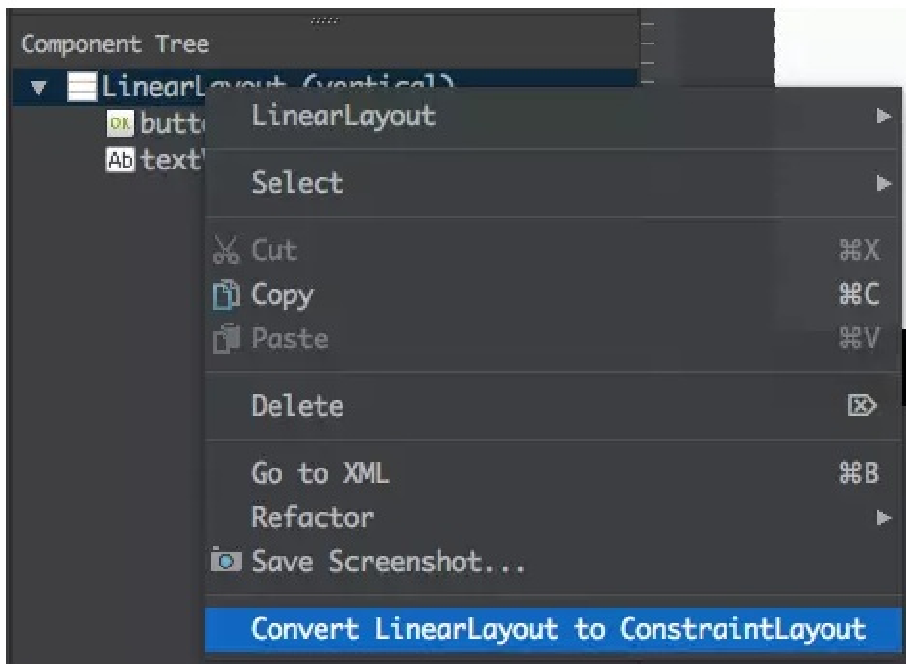


你可以为布局添加水平和竖直引导线，针对这条线来作为基准线布局，如图所示：



11.ConstraintLayout布局转换

通过Android Studio，我们可以很方便的把一个普通布局转化为ConstraintLayout，在布局设计器的左边栏下面的Component Tree来进行转换，如图所示：



转换还是很赞的，但目前还没试过复杂的布局是否有问题。

12.从代码角度理解**ConstraintLayout**属性

ConstraintLayout被称为增强的RelativeLayout，是有它的原因的，相对布局提供了layout_toBottomOf类似这样的属性来控制组件间的相对位置，那么ConstraintLayout实际上也是一样的，我们来看这样一个属性：

```
app:layout_constraintTop_toBottomOf
```

他代表的是『期望组件的顶部，与指定组件的底部对齐』，那么了解了这个解释方式，其它的属性就很好理解了，所以说，虽然ConstraintLayout不太建议通过代码来布局了，但能理解代码的含义，对理解ConstraintLayout布局是非常有帮助的。

第四章 管理

TODO

未完成的任务或者是需要解决的bug，都可以使用TODO来标识，用来提醒团队成员需要注意这些地方，Android Studio 对TODO 提供了非常好的支持，建议掌握此工具。

增加TODO

增加TODO 只需要在注释中添加TODO 关键字就可以了。

> 有一种比较方便的方法就是，直接在代码里输入小写的todo,Android Studio 会自动生成注释并且加上日期，类似 "\/\/ *TODO: 2016\/9\/16* "

操作步骤:

添加注释 —> 输入TODO —>添加说明

> Java、xml、gradle 文件都支持TODO

查看TODO任务

TODO工具窗口中查看TODO任务

操作步骤:

菜单栏 —> View —> TODO

TODO间上下跳转

光标定位在某个TODO任务上,可以使用快捷键快速上下跳转来查看

快捷键:

Mac: Alt + Command + 向上箭头 和 Alt + Command + 向下箭头

Windows\/Linux: Ctrl + Alt + 向上箭头 和 Ctrl + Alt + 向下箭头

设置TODO

设置TODO界面可以从偏好设置中打开,也可以从TODO工具窗口打开.虽然路径不同,但设置项是相同的. 我们下面均以偏好设置中的设置为例进行讲解.

设置TODO关键字匹配区分大小写

默认TODO关键字匹配是不区分大小写的,因此不管是TODO还是todo都被识别为TODO任务，如果想要区分大小写，需要设置。

操作步骤:

Preferences —> Editor —> TODO —> 在需要区分大小写的匹配前面勾选Case Sensitive。

勾选Case Sensitive之后确定设置.。再查看之前的todo就不会被识别为TODO任务了。

自定义TODO关键字

如果想要通过TODO来分配任务, 使用自定义关键字会更加方便管理。

比如: todo-quanke是分配给我自己的任务,todo-zhiran是分配给志然的任务。

操作步骤:

Preferences —> Editor —> TODO —> 添加按钮 —>输入匹配表达式 —> 选择icon —> 设置颜色

第五章 编辑

复制、粘贴、选择、查找、替换应该是我们在编写代码时最常用的操作了，Android Studio可以让这些操作变得简单和高效。

复制 /粘贴 /剪切 /撤销/重做

复制

菜单栏: Edit —> Copy

快捷键:

Mac: Command + C

Windows/Linux: Ctrl + C

复制为纯文本

操作步骤:

菜单栏 —> Edit —> Copy as Plain Text

复制引用

操作步骤:

菜单栏 —> Edit —> Copy Reference

快捷键:

Mac: Alt + Command + Shift + C

Windows/Linux: Ctrl + Alt + Shift + C

复制行

快速复制一行内容

操作步骤:

菜单栏: Edit —> Duplicate Line

快捷键:

Mac: Command + D

Windows/Linux: Ctrl + D

粘贴

菜单栏: Edit —> Paste

快捷键:

Mac: Command + V

Windows/Linux: Ctrl + V

从复制历史中选择粘贴

操作步骤:

菜单栏 —> Edit —> Paste from History

设置粘贴历史记录的个数

Android Studio默认允许记录5个粘贴历史, 但这个是可以设置的，如果你想调整粘贴历史记录，可以这样设置.

操作步骤:

Preferences —> Editor —> 点击General —> 在Limits中调整Maximum number of contents to keep in clipboard的个数

剪切

菜单栏: Edit —> Cut

快捷键:

Mac: Command + X

Windows/Linux: Ctrl + X

撤消：

菜单栏: Edit —> Undo

快捷键:

Mac: Command + Z

Windows/Linux: Ctrl + Z

重做

菜单栏: Edit —> Redo

快捷键:

Mac: Shift + Command + Z

Windows/Linux: Ctrl + Shift + Z

查找替换

查找工具栏

快捷键

Mac: Command + F

Windows\/Linux: Ctrl + F

快速查找

先选中复制,然后调出查找工具栏,选中部分就被自动填入搜索框，按回车键快速搜索。

操作步骤（快捷键）:

Command + C (复制单词)

Command + F (调出查找工具栏)

Enter (回车开始查找)

在查找结果中跳转

下一个匹配到的结果

快捷键:

Mac: Command + G

Windows\/Linux: F3

上一个匹配到的结果

快捷键:

Mac: Command + Shift + G

Windows\/Linux: Shift + F3

在匹配到的相同的结果间跳转

快捷键:

Mac\/Windows\/Linux: Enter

选择查找结果

同时选中所有查找到的结果

操作步骤:

菜单栏: Edit —> Find —> Select All Occurrences

快捷键:

Mac: control + Command + G

Windows\/Linux: Ctrl + Alt + Shift + J

连续选择当前及下一个结果

操作步骤:

菜单栏: Edit —> Find —> Add Select for Next Occurrence

快捷键:

Mac: control + G

Windows\/Linux： Alt + J

一个一个取消选择

操作步骤:

菜单栏: Edit —> Find —> UnSelect Occurrence

快捷键:

Mac: control + Shift + G

Windows\/Linux： Alt + Shift + J

指定查找路径

操作步骤:

菜单栏: Edit —> Find —> Find in Path...

快捷键:

Mac: Shift + Command + F

Windows\/Linux: Ctrl + Shift + F

替换

操作步骤:

菜单栏: Edit —> Find —> Replace

快捷键:

Mac: Command + R

Windows\/Linux: Ctrl + R

指定替换路径

操作步骤:

菜单栏: Edit —> Find —> Replace

快捷键:

Mac: Command + Shift + R

Windows\/Linux: Ctrl + Shift + R

指定查找范围:

操作步骤:

点击Find —> 弹出确认对话框

> 默认点击Replace是一个一个替换的,你也可以有其它选择：

- 点击Skip会跳过当前匹配结果
- 点击Replace All in This File会替换当前文件中所有的匹配结果
- 点击Skip To Next File会跳到下一个文件
- 点击All Files就全部替换啦
- 点击Review,在工具栏显示查找结果,点击后可以查看详情.

> Replace Selected替换选匹配项,Replace All全部替换.

查找替换历史:

Android Studio会记录你所有的查找和替换的历史和配置,当你再一次进行查找的时候默认是使用上一次的替换记录

在结构中搜索和替换

操作步骤:

菜单栏: Edit —> Find —> Search Structurally\/Replace Structurally —>打开 [Search Structurally]

查找用法

看某个方法或变量在哪些地方使用

操作步骤:

菜单栏: Edit —> Find —> Find Usages

右键菜单: Find Usages

快捷键:

Mac: fn + option + F7

Windows\/Linux: Alt + F7

设置查找用法的过程和范围

操作步骤:

菜单栏: Edit —> Find —> Find Usages Settings

快捷键:

Mac: option + Command + Shift + F7

Windows\/Linux: Ctrl + Alt + Shift + F7

大小写替换

操作步骤:

菜单栏: Edit —> Toggle Case

快捷键:

Mac: Shift + Command + U

Windows/Linux: Ctrl + Shift + U

Macros（宏）

什么是宏？

是一种批量批处理的称谓，它是一些命令组织在一起，作为一个单独命令完成一个特定任务。

录制回放宏

需求: 自动保存代码,然后运行代码

第一步: Edit --> Macros --> Start Macros Recording --> Android Studio右下角显示开始录制提示

第二步: 按下command + s 保存文件

第三步: 按下 control + s 运行代码

第四步: Edit --> Macros --> Stop Macros Recording --> 指出输入框,输入已录制的宏的名字

第五步: 定义快捷键

Android Studio --> Keymap -- Macros -- 找到我们刚才录制的宏 [AutoSaveAndRun]

编辑宏

Edit --> Macros --> Edit Macros

使用列选择模式

操作步骤:

菜单栏 —> Edit -- > Column Selection Mode (列选择模式)

快捷键:

Mac: Shift + Command + 8

Windows/Linux: Alt + Shift + Insert

扩大/缩小选择范围

执行一次扩大选择,选择范围就会相应的扩大

操作步骤:

菜单栏: Edit --> Extend Selection

快捷键:

Mac: option + ↑

Windows/Linux: Ctrl + W

缩小选择范围与上面的扩大选择相反

操作步骤:

菜单栏: Edit --> Shrink Selection

快捷键:

Mac: option + ↓

Windows/Linux: Ctrl + Shift + W

合并两行内容

可以智能的合并字符串、注释、声明和赋值

操作步骤:

菜单栏: Edit —> Join Lines (合并行)

快捷键:

Mac: control + shift + j

Windows/Linux: Ctrl + Shift + J

自动补全当前的语句

自动补全当前的语句可以帮你完成正在输入的语句的剩余部分，自动增加漏掉的大括号小括号和必要的格式化处理。

操作步骤:

菜单栏: Edit —> Complete Current Statement (自动补全当前的语句)

快捷键:

Mac: shift + command + 回车 Windows/Linux: Ctrl + Shift + 回车

缩排

操作步骤:

菜单栏: Edit —> Indent Selection (必须选中行)

快捷键：

Mac/Windows/Linux: Tab

取消缩进

操作步骤:

菜单栏: Edit —> UnIndent Line or Selection

快捷键：

Mac/Windows/Linux: Shift + Tab

切换大小写字母

操作步骤:

菜单栏: Edit —> Toggle Case

快捷键:

Mac: Shift + Command + U

Windows/Linux: Ctrl + Shift + U

第六章 窗口视图

导航

IDE里的导航就是，在各个类、文件、方法中快速跳转，熟悉运用导航快捷键，可以大大提高工作效率

搜索并打开某个类文件

如果需要搜索类名，需要知道类的名字，可以支持模糊匹配

操作步骤：

菜单栏：Navigate-->Class

快捷键:

Mac: Command + O

Windows\/Linux: Ctrl + N

> 精确跳转到类文件的某一行

精确跳转到类文件的某一行

和搜索某一个类文件一样，只是在后面加上`:`和行号

例如：EmptyLayout:50

搜索并打开某个文件

如果需要搜索某个文件 ，需要知道文件的名字，可以支持模糊匹配

操作步骤：

菜单栏：Navigate-->File

快捷键:

Mac: Shift + Command + O

Windows\/Linux: Ctrl + Shift + N

搜索并打开某个文件或方法

此方法支持文件和方法搜索

操作步骤：

菜单栏：Navigate-->Symbol

快捷键:

Mac: Option + Command + O

Windows\/Linux: Ctrl + Alt + Shift + N

快速跳转到某一行

操作步骤：

菜单栏: Navigate --> Line

快捷键**:**

Mac: Command + L

Windows\/Linux: Ctrl + G

使用自定义代码块

在代码片段的开始和结束加上表示，有了这个表示以后，就可以展开和折叠

自定义代码库如下所示

```
//<editor-fold desc="Description">
findViewById(R.id.btnLoading).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        emptyLayout.showLoading();
    }});
findViewById(R.id.btnEmpty).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        emptyLayout.showEmpty();
    }});
//</editor-fold>
```

快捷键:

Windows\/Linux: Ctrl + Shift + T 之后选择 `<editor-fold...> `

在自定义代码块之间选择和跳转

操作步骤：

菜单栏: Navigate --> Custom Folding

快捷键:

Mac: Option + Command + .

Windows\/Linux: Ctrl + Alt + .

> 注意：不要使用小写键盘的 .

快速跳转到光标的历史位置

快速跳转到光标上一个历史位置

操作步骤:

菜单栏: Navigate —> Back

快捷键:

Mac: Command +〔

Windows\/Linux: Ctrl + Alt + ←

快速跳转到光标下一个历史位置

操作步骤:

菜单栏: Navigate —> Forward

快捷键:

Mac: Command +〔

Windows\/Linux: Ctrl + Alt + →

快速跳转到编辑过的位置

光标快速跳转到上一个编辑过的位置

操作步骤:

菜单栏: Navigate —> Last Edit Location

快捷键:

Mac: Shift + Command + delete

Windows\/Linux: Ctrl + Shift + Backspace

光标快速跳转到下一个编辑过的位置

操作步骤：

菜单栏: Navigate —> Next Edit Location

快捷键:

Mac:

Windows\/Linux:

标记书签

Android Studio 书签是什么？

在Android Studio中的书签是指对代码的标记，标记我们阅读的位置.。给代码加上书签, 就可以快速的在不同的书签中切换。

标记书签

前提条件：

光标放到需要标记的位置

操作步骤：

菜单栏: Navigate —> Bookmarks —> Toggle Bookmark

快捷键：

Mac: Fn + F3

Windows\/Linux: F11

第1步: 把光标定位在需要加书签的那一行代码

第2步: 快捷键(Mac): Fn + F3

结果: 这行代码就会打上了标签

> 取消书签和标记书签操作时一样的

使用助记符标书签

使用一个符号帮助我们记忆

前提条件：

光标放到需要标记的位置

操作步骤:

菜单栏: Navigate —> Bookmarks —> Toggle Bookmark with Mnemonic

快捷键：

Mac: Fn + Option + F3

Windows\/Linux: Ctrl + F11

实例演示:

第1步: 把光标定位在想加书签的那一行代码

第2步: 快捷键(Mac): Fn + Opetion + F3

第3步: 在弹出的Bookmark Mnemonic弹窗中选择助记符,助记符为一个数字和字母

第4步: 然后在编辑器的左边栏就会显示助记符书签

把光标放到助记符书签上,会提示你这是一个书签.

> 取消书签和标记书签操作时一样的

管理书签

对书签进行搜索、排序、修改描述、删除、查看等操作

打开书签管理界面

操作步骤:

菜单栏: Navigate —> Bookmarks —> Show Bookmarks

快捷键：

Mac: Fn + Command + F3

Windows\/Linux: Shift + F11

修改书签描述

给书签加上描述信息

操作步骤:

在书签管理界面点击左上角的编辑按钮 —> 在弹窗中输入描述

快捷键：

Mac: Command + 回车 —> 在弹窗中输入描述

Windows\/Linux: Ctrl + 回车 —> 在弹窗中输入描述

在收藏夹中管理书签

操作步骤:

工具栏 —> Favorites —>Bookmarks

快捷键：

Mac: Alt + 2 --> 选择 Bookmarks

Windows\/Linux: Alt + 2 --> 选择 Bookmarks

快速跳转到导航栏

写代码的时候想快速跳转到导航栏,在导航栏进行文件切换时使用

操作步骤:

菜单栏: Navigate —> Jump to Navigation Bar

快捷键:

Mac: Command + ↑

Windows\/Linux: Alt + Home

快速跳转到声明

在写代码或阅读代码时,如果需要查看一个引用首次被声明的位置,可以使用此功能.

你可以在代码中的任何位置查看一个引用首次被声明的位置.

> 此功能使用频率非常高，建议熟练掌握

操作步骤:

菜单栏: Navigate —> Declaration

快捷键:

Mac: Command + B

Windows\/Linux: Ctrl + B

鼠标：

Mac: 按住Command键 —> 点击类、变量或方法

Windows\/Linux: 按住 Ctrl 键 —> 点击类、变量或方法

快速查看某个引用的类型声明

操作步骤:

菜单栏: Navigate —> Type Declaration

快捷键:

Mac: Shift + Command + B

Windows\/Linux: Ctrl + Shift + B

鼠标：

Mac: 按住Command+Shift键 —> 点击类、变量或方法

Windows\/Linux: 按住 Ctrl +Shift键 —> 点击类、变量或方法

快速跳转到实现

查看某个接口在哪里被实现

操作步骤:

菜单栏: Navigate —> Implementation

快捷键:

Mac: Option + Command + B

Windows\/Linux: Ctrl + Alt + B

快速跳转到父类

查看某个子类的父类或者快捷跳转到父类

前提条件:

光标定位在方法体内或类名上.

操作步骤:

菜单栏: Navigate —> Super Method

快捷键:

Mac: Command + U

Windows\/Linux: Ctrl + U

在快速类和被测试类之间快速跳转

> 相对来说使用频率不高，暂时不编写，以后加上

查看相关联的文件

比如Activity 绑定了xml，可以使用此功能快速跳转到绑定的xml

操作步骤:

菜单栏: Navigate —> Related Symbol

快捷键:

Mac: control + command + ↑

Windows\/Linux: Ctrl + Alt + Home

查看文件结构

快速调出当前文件的结构，并通过模糊匹配快速跳转至指定的方法

> 此功能很实用，建议熟练掌握

操作步骤:

菜单栏: Navigate —> File Structure

快捷键:

Mac: Fn + Command + F12

Windows\/Linux: Ctrl + F12

显示匿名内部类

勾选 Show Anonymous Classes

显示所有继承的方法

勾选 Show inherited members

输入时缩小

勾先输入时缩小(Narrow down on tying) ：搜索结果会自动过滤,不匹配的不会展示

不勾先输入时缩小(Narrow down on tying):不匹配的项是不会被过滤掉，会展示

查看类的层次结构图

操作步骤:

菜单栏: Navigate —>Type Hierarchy

快捷键:

Mac: control + H

Windows\/Linux: Ctrl + H

查看方法类型的层次结构图

查看某个方法在哪个类中定义,在哪些类中实现

操作步骤:

菜单栏: Navigate —> Method Hierarchy

快捷键:

Mac: Shift + Command + H

Windows\/Linux: Ctrl + Shift + H

查看方法调用层次结构

操作步骤:

菜单栏: Navigate —> Call Hierarchy

快捷键:

Mac: control + option + H

Windows\/Linux: Ctrl + Alt + H

快速跳转到错误代码位置

代码如果出现错误,编辑器会在右边栏高亮显示红色的条标,如果有多处错误就会显示多个条标.

如果想快捷查看错误代码,可以通过点击右边栏的条标进行快速跳转。

跳转到下一个错误位置

操作步骤:

菜单栏: Navigate —> Next Highlighted Error

快捷键:

Mac: Fn + F2

Windows\/Linux: F2

跳转到上一个错误位置

操作步骤:

菜单栏: Navigate —> Previous Highlighted Error

快捷键:

Mac: Fn + Shift + F2

Windows\/Linux: Shift + F2

方法间前后跳转

跳转到下一个方法

操作步骤:

菜单栏: Navigate —> Next Method(下一个方法)

快捷键:

Mac: control + ↑ (此快捷键与可能与系统快捷键冲突,请自行修改)

Windows\/Linux: Alt + ↑

跳转到上一个方法

操作步骤:

菜单栏: Navigate —> Previous Method(上一个方法)

快捷键:

Mac: control + ↓ (此快捷键与可能与系统快捷键冲突,请自行修改)

Windows\/Linux: Alt + ↓

使用翻页

向上翻页:

快捷键：

Mac： Fn + ↑

Windows\/Linux:PgUp

向下翻页:

快捷键：

Mac： Fn + ↓

Windows\/Linux:PgDn

选择当前文件在哪里显示

操作步骤:

菜单栏: Navigate —> Select in

快捷键:

Mac: Fn + Option + F1

Windows\/Linux: Alt + F1

光标快速跳转到编辑器

如果编辑器处于活跃状态(文件处于打开状态), 当焦点不在编辑器时, 按下ESC键可以让焦点从任何其他工具窗口返回到活跃的编辑器.

按下ESC键仅光标跳转到编辑器, Shift + ESC 关闭工具窗口后再跳转到编辑器。

第八章 编码

Android Studio中提供的代码生成、活动模板、自动补全和格式化代码的功能，能够帮助我们极大的提高的编码效率。

覆写或者实现方法

覆写父类的方法或实现接口方法

操作步骤:

菜单栏: Code —> Override Method

快捷键:

Mac: control + O

Windows\/Linux: Ctrl + O

实例演示:

实现接口方法

Override Method可以实现接口方法也可以覆写父类的方法，但Implement Methods只能实现接口方法

操作步骤:

菜单栏: Code —> Implement Methods

快捷键:

Mac: control + L

Windows\/Linux: Ctrl + I

实现代理方法

生成构造函数

操作步骤:

菜单栏: Code —> Generate —> Constructor

快捷键:

Mac: command + N

Windows\/Linux: Alt + Insert —> Constructor

生成Getter和Setter方法

操作步骤:

菜单栏: Code —> Generate —> Getter and Setter

快捷键:

Mac: command + N

Windows\/Linux: Alt + Insert —> Getter and Setter

覆写equals和hashcode方法

操作步骤:

菜单栏: Code —> Generate —> equals() and hashCode()

右键菜单:Generate —> equals() and hashCode()

快捷键:

Mac: command + N

Windows\/Linux: Alt + Insert —> equals() and hashCode()

快速覆写toString方法

操作步骤:

菜单栏: Code —> Generate —> toString()

右键菜单: Generate —> toString()

快捷键:

Mac: command + N

Windows\/Linux: Alt + Insert —> toString()

插入版权信息

操作步骤:

菜单栏: Code —> Generate —> Copyright

右键菜单: Generate —> Copyright

快捷键:

Mac: command + N

Windows\/Linux: Alt + Insert —> Copyright

> 如果你当前没有定义任何Copyright信息,会提示设置版权信息，点击OK，进去版权信息配置

提取或删除语句

从for, foreach, if..elseif...else, try...catch...finally, while...do, do...while中快速提取或删除代码

> 提取或删除代码的时候,被绿色选中的是要提取的内容,被红色选中的是要被删除的内容.

操作步骤:

菜单栏: Code —> Unwrap\/Remove

快捷键:

Mac: command + shift + Delete

Windows\/Linux: Ctrl + Shift + Delete

智能自动补全会将不适用的条目过滤掉，只显示可用的类、变量、属性或者方法

操作步骤:

菜单栏:Code —> Completion —>SmartType

快捷键:

Mac: control + Shift + 空格

Windows\/Linux: Ctrl + Shift + 空格

代码补全

当弹出代码补全提示时,可以鼠标|Enter|Tab|!V.|;进行补全,但这几种补全方法功能是不一样的。

补全后不会删除后面的代码

操作步骤:

弹出代码补全提示—> 光标选中补全代码—> 鼠标VEnter

补全后删除后面的代码

操作步骤:

弹出代码补全提示—> 光标选中补全代码—>Tab

布尔值取反补全

操作步骤:

弹出布尔值代码补全提示—> 光标选中补全代码（是弹出代码提示时候的选择）—> 叹号(!)

循环扩展词

展开或折叠代码

展开或折叠代码

操作步骤:

菜单栏:Code —> Folding —> Expand

快捷键:

Mac: command + “+”

Windows\/Linux: Ctrl + "+"

展开或折叠代码

操作步骤:

菜单栏:Code —> Folding —> Collapse

快捷键:

Mac: command + “-”

Windows\/Linux: Ctrl + "-"

展开或折叠当前代码块中的所有子模块

操作步骤:

菜单栏:Code —> Folding —> Expand Recursively

快捷键:

Mac: option + command + “+”

Windows\/Linux: Ctrl + Alt + "+"

折叠当前代码块中的所有子模块

操作步骤:

菜单栏:Code —> Folding —> Collapse Recursively

快捷键:

Mac: option + command + “-”

Windows\/Linux: Ctrl + Alt + NumPad + "-"

展开全部代码块

操作步骤:

菜单栏:Code —> Folding —> Expand All

快捷键:

Mac: shift + command + "+"

Windows\/Linux: Ctrl + Shift + "+"

折叠全部代码块

操作步骤:

菜单栏:Code —> Folding —> Collapse All

快捷键:

Mac: shift + command + "-"

Windows\/Linux: Ctrl + Shift + "-"

展开和折叠当前文件中的所有注释

> 使用此功能会自动展开当前文件里的所有注释,不会展开代码。

展开代码注释: Code —> Folding —> Expand doc comments

折叠代码注释: Code —> Folding —> Collapse doc comments

指定展开层级

指定展开代码层级: Code -- Folding -- Expand to level

指定全部展开代码的层级: Code -- Folding -- Expand all to level

折叠\/展开选中区域

选中要折叠代码片段.

操作步骤:

菜单栏:Code —> Folding —> Fold Selection \/ Remove regio

快捷键: Mac: command + . | Windows\/Linux: Ctrl + .

> 选中的代码就会被折叠，再触发一次，代码就会被展开.

折叠代码片段

此功能用来折叠{}中的代码片段和自定义的代码片段.

操作步骤:

菜单栏:Code —> Folding —> fold code block

快捷键:

Mac: shift + command + .

Windows\/Linux: Ctrl + Shift + .

代码模板

Live Template（ 实时模板 ）

Live Template就是把常用的代码提取成一个模板,在编写代码的时候可以通过缩写调出这个模板,达到快速输入,提高效率的目的.

Android Studio中提供的Live Template定义了一些常用的缩写，我们可以通过输入缩写快速生成常用的代码模板。

操作步骤:

菜单栏:Code —> Insert Live Template

快捷键:

Mac: command + J

Windows\/Linux: Ctrl + J

然后在弹出的列表中选取缩写。

当然也可以:

直接输入LiveTemplate定义的缩写,然后按下Tab键插入。

常用缩写

```
Log.d("");
```

使用代码模板包围

操作步骤:

菜单栏:Code —> Surround with Live Template

快捷键:

Mac: option + command + J

Windows\/Linux: Ctrl + Alt + J

然后选择代码模板

在Select Template弹窗中点击模板或输入模板前面的大写字母都可以插入代码模板.

编辑Live Templates

操作步骤：

菜单栏: File->Settings -> Editor -> Live Templates

注释行

操作步骤:

菜单栏: Code —> Comment with Line Comment

快捷键:

Mac: command + \/

Windows\/Linux: Ctrl + \/

注释代码块

操作步骤:

菜单栏: Code —> Comment with Block Comment

快捷键:

Mac: option + command + \/

Windows\/Linux: Ctrl + Shift + \/

格式化代码

操作步骤:

菜单栏: Code —> Reformat Code

右键菜单: Reformat Code

快捷键:

Mac: option + command + L

Windows\/Linux: Ctrl + Alt + L

配置格式化选项

通过快捷键(Mac): option + command + shift + L 打开配置对话框

自动缩进行

操作步骤:

菜单栏: Code —> Auto-Indent Lines

快捷键:

Mac: option + control + I

Windows\/Linux: Ctrl + Alt + I

删除多余导入

当有多余包的导入的时候，可以使用此方法去掉多余导入

操作步骤:

菜单栏: Code —> Optimize Imports

快捷键:

Mac: control + option + O

Windows\/Linux: Ctrl + Alt + O

右击文件菜单:Optimize Imports

重新排列代码

操作步骤:

菜单栏:Code —> Rearrange Code

向上移动代码

如果要移动一段代码, 需要先选中这段代码.

如果要移动方法, 光标需要定位在方法名上.

操作步骤:

菜单栏: Code —> Move Statement Up

快捷键:

Mac: shift + command + ↑

Windows\/Linux: Ctrl + Shift + ↑

向下移动代码

如果要移动一段代码, 需要先选中这段代码.

如果要移动方法, 光标需要定位在方法名上.

操作步骤:

菜单栏:Code -- Move Statement Down

快捷键:

Mac: shift + command + ↑ |

Windows\/Linux: Ctrl + Shift + ↑

第九章 检查

第十章 重构

重构，是指在不改变外部行为的条件下，对现有的代码进行改善，以增加可读性，使设计和逻辑更加清晰。Android Studio的重构功能可以帮助我们安全、简单、快速的完成代码重构。

重命名

操作步骤:

菜单栏: Refactor --> Rename

快捷键：

Mac: fn + Shift + F6

提炼方法

提炼方法 （ Extract Method），把一段代码提取出来作为一个单独的方法使用。

操作步骤:

菜单栏: Refactor —> Extract —> Method

快捷键：

Mac: option + Command + M

提炼方法对象

操作步骤:

菜单栏: Refactor —> Extract —> Method Object...

更改方法签名

改变方法的名称,改函数的可访问性,对参数进行添加、删除、重命名和重新排序。

操作步骤：

菜单栏: Refactor --> Change Signature

快捷键：

Mac: fn+ command + F6

迁移变量类型（ Type Migration ）

改变变量的类型。

操作步骤:

菜单栏: Refactor --> Type Migration

快捷键

Mac: fn+ Shift + Command + F6

转成静态方法

操作步骤:

菜单栏: Refactor-->Make Static

静态方法转为实例方法

操作步骤:

菜单栏: Refactor --> Convert to Instance Method

移动类

把一个类移动到不同的包下

> 可以直接把类托动到目标地址

操作步骤:

菜单栏: Refactor —> Move...

快捷键

Mac: fn + F6 (可能会跟系统快捷键冲突)

移动静态方法

操作步骤:

菜单栏: Refactor —> Move...

快捷键：

Mac: fn + F6 (可能会跟系统快捷键冲突)

移动静态字段

操作步骤:

菜单栏: Refactor —> Move...

快捷键：

Mac: fn + F6 (可能会跟系统快捷键冲突)

安全删除

为了保证安全的删除, Android Studio会找到被删除字符, 结果会在工具栏展示,可以在删除之前对代码进行修正。

操作步骤:

菜单栏: Refactor —> Safe Delete...

提炼为变量

将一个表达式提炼为一个变量,并使用变量替换原来的表达式.

操作步骤:

菜单栏: Refactor —> Extract —> Variable...

快捷键:

Mac: option + Command + V

提炼为常量

临时变量快速提炼出静态常量

操作步骤:

菜单栏: Refactor —> Extract —> Constant...

快捷键:

Mac: option + Command + C

提炼字段

将选中的表达式赋值给一个新声明的字段，原来的表达式被新的字段代替。

操作步骤:

菜单栏: Refactor —> Extract —> Field...

快捷键：

Mac: option + Command + F

提炼参数

在方法中添加新的参数,且更新这个方法的调用。

操作步骤:

菜单栏: Refactor —> Extract —> Parameter...

快捷键 :

Mac: option + Command + P

提炼委托

提炼委托重构是在一个类中提炼一些字段和方法到一个单独的、新创建的类中。

当一个类变得太大并且做了太多的事情时, 我们就需要考虑重构, 将这个类拆分为更小、更有凝聚力的类。

操作步骤:

菜单栏: Refactor —> Extract —> Delegate...

提炼接口

提炼接口重构是从一个已存在的类中提炼接口。

操作步骤:

菜单栏: Refactor —> Extract —> Interface ...

提炼父类

操作步骤:

菜单栏: Refactor —> Extract —> Superclass...

内联方法

操作步骤:

菜单栏: Refactor —>Inline

快捷键:

Mac: option+Command + N

内联临时变量

操作步骤:

菜单栏: Refactor —>Inline

快捷键:

Mac: option+Command + N

查找并替换重复代码

操作步骤:

菜单栏: Refactor —> Find and Replace Code Duplicates...

反转布尔值

把一个Boolean方法或变量改变为相反的意义,也就是原来是true的变为false，原来是false的变为true。

操作步骤:

菜单栏: Refactor —> Invert Boolean

把成员拉到父类

操作步骤:

菜单栏: Refactor —> Pull Members Up...

把成员推到子类

操作步骤:

菜单栏: Refactor —> Push Members Down...

尽可能使用接口

操作步骤:

菜单栏: Refactor —> Use Interface Where Possible...

使用委托替换继承

操作步骤:

菜单栏: Refactor —> Replace Inheritance with Delegation...

移除中间人

操作步骤:

菜单栏: Refactor —> Remove Middleman...

包装方法返回值

使用一个新建的包装类来替换一个方法的返回值, 这个类也可以是已存在的类.

操作步骤:

菜单栏: Refactor —> Wrap Return Value...

将匿名类转成内部类

把一个匿名类转成内部类

操作步骤:

菜单栏: Refactor —> Convert Anonymous to Inner...

封装字段

操作步骤:

菜单栏: Refactor —> Encapsulate Field ...

使用查询替换临时变量

操作步骤:

菜单栏: Refactor —> Replace Temp with Query...

使用工厂方法替换构造方法

操作步骤:

菜单栏: Refactor —>Replace Constructor with Factory Method...

泛型化

将那些没有使用泛型的代码转化为泛型可识别的代码

操作步骤:

菜单栏: Refactor —> Gentrify...

迁移

把项目中使用的老包和类切换为新包

操作步骤:

菜单栏: Refactor —> Migrate...

第十一章 打包构建（Gradle）

在Android Studio上，打包运行，导入的包管理，都是使用Gradle，所以了解 Gradle 是非常必要的。这章我们一起学习 Gradle。

Gradle是什么？

Gradle是一个自动化构建工具,采用Groovy的Domain Specific Language（领域特定语言）来描述和控制构建逻辑.它的特点是语法简洁、可读性强、配置灵活等等. 基于Intellij IDEA社区版本开发的Android Studio天生支持Gradle构建程序.

Gradle使用指南请看:https:\/\/docs.gradle.org\/current\/userguide\/userguide.html

Groovy是什么?

Groovy是一种基于JVM的敏捷开发语言,它结合了Python、Ruby、Smalltalk的许多强大特性. Groovy 代码既能够与 Java 代码很好的结合,也能够用于扩展现有的代码.

更多Groovy请参考:http:\/\/www.groovy-lang.org\/

Gradle的特点:

- Gradle支持多工程构建和局部构建.
- Gradle支持远程或本地依赖管理: 支持从远程maven仓库、nexus私服、ivy仓库以及本地仓库获取依赖.
- Gradle与Ant、Maven兼容.
- Gradle可轻松迁移: Gradle适用于任何结构的工程.我们可以在同一个开发平台平行构建原工程和Gradle工程.
- Gradle使用灵活: Gradle的整体设计是以作为一种语言为导向的,而非成为一个严格死板的框架.
- Gradle免费开源Gradle跟IDE集成的非常好
- Gradle可以更容易地集成到自动化构建系统

Gradle中的Project、Task和Plugin

项目是什么？

项目(project)是指我们的构建产物（比如Jar包）或部署的产物（将应用程序部署到生产环境）,每个项目包含一个或多个任务.

任务是什么?

任务(Task)是指不可分的最小工作单元，代表一个逻辑上较为独立的执行过程,比如:编译、复制、打包.

插件是什么？

Gradle只是提供了构建项目的一个框架,所有有用的特性都由Gradle插件提供，一个Gradle插件能够：

- 向项目中添加新任务.
- 为新加入的任务提供默认配置，这个默认配置会在项目中注入新的约定(如源文件位置).
- 加入新的属性, 可以覆盖插件的默认配置属性.
- 为项目加入新的依赖.

更多请参考: http:\/\/www.gradle.org\/docs\/current\/userguide\/standard_plugins.html

项目、任务和插件的关系是什么？

项目代表要被构建的组件或整个项目,它为任务提供了执行的上下文，而插件用来向项目中添加属性和任务.一个任务可以读取和设置项目的属性以完成特定的操作.

如何配置Gradle构建？

Gradle的构建肯定会包含下面这几个配置文件:

Gradle构建脚本（build.gradle）指定了一个项目和它的任务.

Gradle属性文件（gradle.properties）用来配置构建属性.

Gradle设置文件（gradle.settings）对于只有一个项目的构建而言是可选的,如果我们的构建中包含多于一个项目，那么它就是必须的.

因为它描述了哪一个项目参与构建.每一个多项目构建都必须在项目结构的根目录中加入一个设置文件gradle.settings.

Gradle中依赖的仓库

仓库是什么?

顾名思义,仓库就存放东西的. 放什么东西呢? 简单来说就是存放我们依赖的jar包.

Gradle支持哪些仓库?

- Maven仓库
- Ivy仓库
- 平级目录仓库

如何在构建中加入这些仓库?

Ivy仓库应该用的人不多吧,这里就不多作介绍了,重点放在maven仓库上.

在build.gradle中添加仓库的声明,方法如下:

从Maven仓库中获取依赖

```
repositories {


1.从指定的远程maven仓库中获取依赖

maven {

url "http://maven.helloword.net/repo"

}



2.从指定的本地maven仓库中获取依赖

maven {

url "file:///Users/bixiaopeng/mvn"

}
```

```
3.从中央Maven仓库中获取依赖

mavenCenter()



4.从新的中央远程仓库中获取依赖

jcenter()



5.从本地仓库中获取依赖

mavenLocal()



6.需要认证的库

maven {

credentials {

username 'user'

password 'password'

}

url "http://repo.helloword.com/maven2"

}

}
```

Maven仓库的三种别名

为了更加方便的加入Maven仓库, Gradle为我们提供了3种别名,分别是:

1.mavenCentral()：

表示从maven中央仓库中获取依赖 地址: http:\/\/repo1.maven.org\/maven2

2.jcenter():

jcenter是一个新的远程中央仓库，兼容maven中央仓库，而且性能更优.

gradle默认使用jcenter作为仓库.

jcenter存放在这里:https:\/\/bintray.com\/

3.mavenLocal():

表示从本地Maven仓库中获取依赖。 本地地址: {user.home}\/.m2\/repository

从平级目录仓库中获取依赖

从本地目录中获取依赖,在build.gradle中添加:

```
repositories {

//从当前项目的平级目录lib中获取依赖

    flatDir(dir: 'lib', name: 'libs directory')

//从当前项目的平级目录libA和libB中获取依赖

flatDir {

    dirs 'libA', 'libB'

    name = 'All dependency directories'

    }

}
```

配置Gradle环境

本文以Mac上配置Gradle环境为例进行介绍.

一. 下载gradle

1.1 下载地址: http://gradle.org/gradle-download/

1.2 下载最新版本:gradle-2.13 (当前最新版为2.13)

当然你也可以选择下载某一个历史版本,在页面右边Choose Vesion 中选择下载到本地后解压.

二. 配置环境变量

2.1 我的本地存放路径:/Volumes/warehouse/dev-tools/tools-jars/gradle-2.132.2 编辑.bash_profile： vim ~/.bash_profile2.3 配置环境变量:

```
export GRADLE_HOME=/Volumes/warehouse/dev-tools/tools-jars/gradle-2.13

export PATH=$PATH:$GRADLE_HOME/bin
```

2.3.使配置立即生效: source ~/.brash_profile

三. 查看配置是否成功

```
~$ gradle -v


------------------------------------------------------------

Gradle 2.13

------------------------------------------------------------


Build time: 2016-04-25 04:10:10 UTC

Build number: none

Revision: 3b427b1481e46232107303c90be7b05079b05b1c

Groovy: 2.4.4

Ant: Apache Ant(TM) version 1.9.6 compiled on June 29 2015

JVM: 1.8.0_91 (Oracle Corporation 25.91-b14)

OS: Mac OS X 10.11.2 x86_64
```

如果电脑上不单独配置Gradle环境也没关系，因为Android Studio中使用了Gradle Wrapper,它可以在我们没有安装gradle的时候进行项目构建,下面我们会讲到.

> Windows和Linux上配置Gradle的开发环境方法差不多,这里就不一一介绍了.

Gradle Wrapper

Gradle Wrapper是什么？

Gradle Wrapper可以理解为是对Gradle的一层封装,使用它可以在没有安装Gradle的系统上使用Gradle来构建项目.

如何做到的呢？

Gradle Wrapper通过两个脚本文件实现这一功能,一个是用于windows的批处理文件gradlew.bat,一个是用于Linux和Uninx的Shell脚本文件gradlew.使用Android Studio创建的项目默认为我们生成了Gradle Wrapper的文件结构.

在gradle/wrapper目录下有两个文件: gradle-wrapper.jar和gradle-wrapper.properties gralde-wrapper.properties文件中声明了gradle的版本和下载地址.

```
#Mon Dec 28 10:00:20 PST 2015

distributionBase=GRADLE_USER_HOME

distributionPath=wrapper/dists

zipStoreBase=GRADLE_USER_HOME

zipStorePath=wrapper/dists

distributionUrl=https\://services.gradle.org/distributions/gradle-2.10-all.zip
```

在第一次使用gradlew进行项目构建的时候,Gradle Wrapper会自动下载gralde-wrapper.properties指定的gradle版本.

> 通过gradlew执行gradle构建跟直接使用gradle是一样的. 如果我们想直接使用gradle构建需要先配置环境变量.

初始化构建环境

在第一次使用gradlew进行项目构建的时候,会对构建环境进行初始化，会把Gradle的安装包、插件和相关依赖下载下来. 在Terminal中输入命令,如下：

```
$ ./gradlew clean

Downloading https://services.gradle.org/distributions/gradle-2.10-all.zip

……………………………………………………………………………..这是一个慢长的等待过程.



Unzipping /Users/bixiaopeng/.gradle/wrapper/dists/gradle-2.10-all/3i2gobhdl0fm2tosnn15
g540i0/gradle-2.10-all.zip to /Users/bixiaopeng/.gradle/wrapper/dists/gradle-2.10-all/
3i2gobhdl0fm2tosnn15g540i0

Set executable permissions for: /Users/bixiaopeng/.gradle/wrapper/dists/gradlea-2.4-al
l/3i2gobhdl0fm2tosnn15g540i0/gradle-2.10/bin/gradle

….

8BUILD SUCCESSFUL
```

首次执行命令行会先去下载gradle安装包，过程可能会非常慢,如果你不着急,就耐心等一下吧。要不然就试试下面的方法。

解决Gradle下载太慢的问题

第一步: 首先我们要知道gradle从哪里下载,下载什么版本？

方法1：在gradle-wrapper.properties中查看gradle下载地址和版本

本例中,下载地址是：https:\/\/services.gradle.org\/distributions\/gradle-2.10-all.zip ,版本是2.10.

方法2：去查看所有分发的gradle版本地址:https:\/\/services.gradle.org\/distributions

在这里可以查看到最新的gradle版本，点击可下载.

第二步: 下载完成后放到什么地方？

本例: \/Users\/quanke\/.gradle\/wrapper\/dists\/gradle-2.4-all\/3i2gobhdl0fm2tosnn15g540i0
你的: \/Users\/你的用户名\/.gradle\/wrapper\/dists\/gradle-版本号-all\/一堆随机生成字符串，这一堆字符是随机生成的，每次发现新的可下载的版本的时候就会生成一堆新的字符,如果想让AndroidStudio识别我们的版本，需要先生成这堆字符哦。

可以先打开Android Studio看有没有生成这堆字符串,如果没有你就执行下命令喽.

第三步: 给gradle脚本设置可执行权限

本例: chmod +x \/Users\/quanke\/.gradle\/wrapper\/dists\/gradle-2.4-all\/3i2gobhdl0fm2tosnn15g540i0\/gradle-2.4\/bin\/gradle

你的: chmod +x \/Users\/你的用户名\/.gradle\/wrapper\/dists\/gradle-版本号-all\/一堆随机生成字符串\/bin\/gradle

然后重新打开Android Studio,gradlew命令就可以用啦.

下面我们不单纯的来讲Gradle,而是通过其在Android项目中的应用来一一讲解.

方法二: 设置代理

翻墙以后妈妈再也不用担心我的学习了.

查看和执行Gradle任务

查看当前项目支持哪些Gradle任务

使用.\/gradlew task来查看当前项目支持哪些Gradle任务.

```
FirstApp$ ./gradlew task

:tasks



------------------------------------------------------------

All tasks runnable from root project(所有从项目根目录可运行的任务)

------------------------------------------------------------

Android tasks(Android 任务)

-------------

androidDependencies - 显示项目的Android依赖

signingReport - 显示每个变种版本的签名信息.

sourceSets - 打印出所有在这个项目中定义的source集合.

Build tasks(构建任务)

-----------

assemble - 编译并打出应用程序所有变种版本的包.

assembleAndroidTest -编译并打出所有测试应用的包.

assembleDebug - 编译并打出Debug版本的包.

assembleRelease - 编译并打出Release版本的包.

build - 执行所有检查并编译打包

buildDependents - 检查所有的依赖并编译打包.

buildNeeded - 检查所有的依赖并编译打包.

clean - 删除构建目录
```

```
compileDebugAndroidTestSources

compileDebugSources

compileDebugUnitTestSources

compileReleaseSources

compileReleaseUnitTestSources

mockableAndroidJar - 创建一个适用于单元测试的android.jar版本.

Build Setup tasks (构建设置任务)

-----------------

init - 被始化一个新的Gradle构建.

wrapper - 生成 Gradle wrapper文件

Help tasks(帮助任务)

----------

buildEnvironment - 显示项目根目录中声明的所有构建脚本的依赖

components - 显示项目根目录产生的组件

dependencies - 显示项目根目录中所有依赖的声明.

dependencyInsight - 显示并洞察项目根目录中一个特殊的依赖关系.

help - 显示帮助信息

model - 显示项目根目录的配置模型.

projects - 显示项目根目录中的子项目.

properties - 显示项目根目录的属性.

tasks - 显示从项目根目录可以运行的任务(

有些显示的任务可能属于子项目)

Install tasks(安装任务)

-------------

installDebug - 编译打包并安装Debug版本的包.

installDebugAndroidTest - 编译打包并安装Debug版本的测试包到设备上
```

```
uninstallAll - 卸载所有版本的包.

uninstallDebug - 卸载Debug版本的包.

uninstallDebugAndroidTest - 从设备上卸载Debug版本的Android测试包

uninstallRelease - 卸载Release版本的包.

Verification tasks(验证任务)

------------------

check - 运行所有检查.

connectedAndroidTest - 在已连接的设备上安装所有flavors(渠道包)并运行instrumentation测试

connectedCheck - 在当前已连接的设备上运行设备检测.

connectedDebugAndroidTest - 在已连接的设备上安装并运行Debug版本的测试.

deviceAndroidTest - 在所有提供的设备上安装并运行instrumentation测试.

deviceCheck - 在所有提供的设备和测试服务器上运行设备检测.

lint - 在所有变种版本上运行lint检测.

lintDebug - 在Debug版本上运行lint检测.

lintRelease - 在Release版本上运行lint检测.

test - 在所有变种版本上运行单元测试.

testDebugUnitTest - 在Debug版本上运行单元测试.

testReleaseUnitTest - 在Release版本上运行单元测试.

Other tasks (其它任务)

-----------

jarDebugClasses

jarReleaseClasses

transformResourcesWithMergeJavaResForDebugUnitTest

transformResourcesWithMergeJavaResForReleaseUnitTest

想查看所有任务和更多详情, 请运行gradlew tasks --all

想查看一个任务的更多详情, 请运行gradlew help --task <task>

BUILD SUCCESSFUL
```

```
Total time: 6.717 secs

这个构建可以更快, 请考虑使用Gradle守护: https://docs.gradle.org/2.10/userguide/gradle_daemo
n.html
```

执行Gradle任务

执行命令:

gradle + 任务名称

或

.\/gradlew + 任务名称

注意: Gradle的Android插件提供了四个顶级任务: 打包(assemble)、检测(check)、构建(build)、清理(clean),当我们执行一个顶级任务的时候会同时执行与其依赖的任务.

比如你执行: .\/gradlew assemble

它会把你配置的所有构建类型(Build Types)全部打出来,默认的构建类型是Debug和Release,因此最起码它会执行两个任务:

gradlew assembleDebug

gradlew assembleRelease

如果有其它的构建类型,任务名应该是:

gradlew assemble+构建类型名

另外你还要知道,执行构建(build)任务会执行 检测(check)和打包(assemble)任务.

常用Gradle任务

1.查看gradle版本

```
$ ./gradlew -v
```

2.编译并打出Debug版本的包.

```
./gradlew assembleDebug
```

3.编译并打出Release版本的包.

```
./gradlew assembleRelease
```

4.执行检查并编译打包

```
./gradlew build
```

打出所有Release和Debug的包.

5.删除build目录

```
./gradlew clean
```

会把app下面的build目录删掉

6.编译打包并安装Debug版本的包

```
./gradlew installDebug
```

7.卸载Debug版本的包

```
./gradlew uninstallDebug
```

8.使用-info查看任务详情

```
./gradlew uninstallDebug -info
```

Gradle工具窗口

Gradle工具窗口列出了当前项目和模块中支持的所有Gradle任务和运行配置，以方便我们快速操作.

Tasks:

Tasks列表里的任务跟我们执行.\/gradlew task得到的任务列表是一样的.

把光标放在某个任务上面显示任务的描述信息:

双击任务即可执行.

任务执行结果:

点击Run工具栏左上角的切换按钮, 可以在任务执行模式和文本式之间切换.

```
Run Configurations:
```

Run Configurations列表中列出了项目中执行过的任务配置,这些配置都是执行任务时自动生成的.

想再次执行的时候可以在配置列表中直接选择.

可以在Run\/Debug Configurations中对这些配置进行修改.

构建项目和模块

编译项目

当我们只想对修改过的文件进行编译时就会使用Make进行编译.它可以指定对项目(Project)或模块(Module)进行编译.

编译项目会同时编译当前项目中所有的模块 ,如果代码较多, 编译时间会比较长.如果想编译快一点需要指定编译某个模块.

操作步骤:

菜单栏 --> Build --> Make Project

快捷键:

Mac: Command + F9

Windows\/Linux: Ctrl + F9

> 在启动app之前默认会先执行Make编译项目.

编译模块

前提条件: 选中某个模块

操作步骤:

菜单栏 --> Build --> Make Module '模块名'

设置自动编译项目

前面我们讲了编译项目是对新产生变化的文件进行一次编译,已经编译过的文件就不用重编译了. 所以如果你想加快编译速度,可以设置项目自动编译.

操作步骤:

Android Studio --> Preferences... --> 搜索Compiler --> 勾选Make project automatically

重新构建项目

操作步骤:

菜单栏：Build --> Rebuild Project(重新构建项目)

Make Project跟Rebuild Project的区别

Make Project跟Rebuild Project都是执行相同的两个任务, 那他们的区别是什么?

最明显示的区别就是执行时间, 从上面的例子中我们可以看出,Make Project用时4.5秒,而Rebuild Project用时需要16秒.

为什么Rebuild Project会慢这么多？

因为Rebuild Project是对整个项目进行了重新编译,不管你之前有没有编译过,因此用时会比较长.

而Make Project只是对新产生变化的文件进行一次编译,已经编译过的文件就不用重编译了,所有用时比较少.

清理项目

清理项目会清空output目录下的文件,并重新编译项目.

操作步骤:

菜单栏：Build --> Clean Project(清理项目)

Gradle Script

我们把项目查看模式切换成Android,所有的文件会通过类型进行归类,这个并不是实际在电脑中的文件结构哦，如果想看实际的物理结构请切换到Project.

切换成Android可以查看所有的Gradle Script:

每个文件后面都有一个灰色字体描述:

1.build.gradle: Project构建文件

2.build.gradle: Module构建文件

3.gradle.properties： gradle配置文件

4.proguard-rules.pro: 混淆规则配置文件

5.graddle-wrapper.properties: gradle wrapper属性文件

6.settings.gradle: 构建项目设置文件

7.local.properties: SDK、NDK配置文件

切换到Project模式我们来看看各个文件之间的关系:

项目中有一个app模块,这个模块是一个Android应用程序,它有自己的构建脚本和混淆配置文件.

项目根目录下的脚本文件是针对它所依赖模块的全局配置.

下面我们详细介绍一下每个文件的作用.

Gradlew配置文件:gradle-wrapper.properties

gradle-wrapper.properties 是gradle wrapper的配置文件.

默认的gradle-wrapper.properties文件内容如下:

```
#Mon Dec 28 10:00:20 PST 2015

distributionBase=GRADLE_USER_HOME

distributionPath=wrapper/dists

zipStoreBase=GRADLE_USER_HOME

zipStorePath=wrapper/dists

distributionUrl=https\://services.gradle.org/distributions/gradle-2.10-all.zip
```

一般这个文件是不用动的,除非你想手动指定gradle的版本,可以修改distributionUrl. 也可以在Project Structure —> Project中设置Gradle version.

项目全局配置文件:settings.gradle

默认的settings.gradle文件内容如下:

```
include ':app'
```

settings.gradle是项目的全局配置文件,主要声明一些需要加入构建的模块,本例中只有一个模块:app.

如果新增一个模块需要在这里添加配置,通过Android Studio添加依赖的module会自动在这里添加相应的配置的.

包含多个模块的格式是这样的:

```
include ':app',':other-module-name'
```

如果包含的是某个目录下的模块,格式是这样的:

```
include ':app', ':dir-name:other-module-name'
```

本地属性配置文件:local.properties

默认的local.properties文件内容如下

```
# 这个文件是由Android Studio自动生成的, 不要修改这个文件---如果你改了也会被自动擦除.

# 这个文件不能被添加到版本控制系统中,因为它包含的信息只针对你本地系统的配置.

# 本地的SDK仅被用于gradle构建.

#Fri May 20 16:18:51 CST 2016

ndk.dir=/../sdk/ndk-bundle

sdk.dir=/../sdk
```

我们可以在Project Structure —> SDK Location中设置SDK和NDK的路径.

修改后会同步到local.properties文件中.

Gradle配置文件:gradle.properties

gradle.properties是gradle的配置文件，build.gradle通过读取这个文件配置的参数来进行相应的构建.

默认的gradle.properties文件内容如下:

```
# 设置整个项目的Gradle.

# IDE (e.g. Android Studio) users:

4# 通过IDE设置将会覆盖Gradle settings文件所有的设置.

# 关于如何配置构建环境的更多详情,请访问:

# http://www.gradle.org/docs/current/userguide/build_environment.html

# 指定用于守护进程的JVM参数.

# 该设置对调整内存设置特别有用.

# 默认值: -Xmx10248m -XX:MaxPermSize=256m

# org.gradle.jvmargs=-Xmx2048m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -D
file.encoding=UTF-8

# 配置完毕后, Gradle 将在并行模式下运行.(并行编译)

# 只应在解耦项目中使用此选项.更多详情, 请访问:

# http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled
_projects

# org.gradle.parallel=true
```

默认的gradle.properties文件内容为空,只有一些注释.可以在这里设置Gradle的代理在gradle.properties添加:

```
systemProp.http.proxyHost=Proxy Server

systemProp.http.proxyPort=Proxy port

# 设置代理的用户名和密码,如果没有就不用设置

systemProp.http.proxyUser=User

systemProp.http.proxyPassword=password

# 设置不需要代理的地址,多个地址使用或'|'隔开

systemProp.http.nonProxyHosts=*.baidu.com|*.taobao.com
```

代码混淆规则配置文件:proguard-rules.pro

如果我们想在打包的时候进行代码混淆,就需要在proguard-rules.pro中配置代码混淆规则.

常用的代码混淆规则:

```
# 在这里添加项目的代码混淆规则

# 混淆规则请参考:http://proguard.sourceforge.net/index.html#manual/usage.html

##################### 一般使用默认 #####################

# 不使用大小写混合类名,混淆后的类名为小写

-dontusemixedcaseclassnames

# 混淆第三方库

-dontskipnonpubliclibraryclasses

# 混淆时记录日志,有助于排查错误

-verbose

# 代码混淆使用的算法.

-optimizations !code/simplification/arithmetic,!code/simplification/cast,!field/*,!class/merging/*

# 代码混淆压缩比,值在0-7之间,默认为5.

-optimizationpasses 5

# 优化时允许访问并修改有修饰符的类和类的成员

-allowaccessmodification

##################### 不混淆 #####################

# 这些类不混淆

-keep public class * extends android.app.Activity

-keep public class * extends android.app.Application

-keep public class * extends android.app.Service

-keep public class * extends android.content.BroadcastReceiver
```

```
-keep public class * extends android.content.ContentProvider

-keep public class * extends android.app.backup.BackupAgent

-keep public class * extends android.preference.Preference

-keep public class * extends android.support.v4.app.Fragment

-keep public class * extends android.support.v4.app.DialogFragment

-keep public class * extends com.actionbarsherlock.app.SherlockListFragment

-keep public class * extends com.actionbarsherlock.app.SherlockFragment

-keep public class * extends com.actionbarsherlock.app.SherlockFragmentActivity

-keep public class * extends android.app.Fragment

-keep public class com.android.vending.licensing.ILicensingService

# Native方法不混淆

-keepclasseswithmembernames class * {

native <methods>;

}

# 自定义组件不混淆

-keep public class * extends android.view.View {

public <init>(android.content.Context);

public <init>(android.content.Context, android.util.AttributeSet);

public <init>(android.content.Context, android.util.AttributeSet, int);

public void set*(...);

}

# 自定义控件类和类的成员不混淆(所有指定的类和类成员是要存在)

-keepclasseswithmembers class * {

public <init>(android.content.Context, android.util.AttributeSet);

}

# 同上

-keepclasseswithmembers class * {
```

```
public <init>(android.content.Context, android.util.AttributeSet, int);

}

# 自定义控件类不混淆

-keepclassmembers class * extends android.app.Activity {

public void *(android.view.View);

}

# 枚举类不被混淆

-keepclassmembers enum * {

public static **[] values();

public static ** valueOf(java.lang.String);

}

# android.os.Parcelable的子类不混淆

-keep class * implements android.os.Parcelable {

public static final android.os.Parcelable$Creator *;

}

# 资源类不混淆

-keepclassmembers class **.R$* {

public static <fields>;

}

##################### 第三方库不混淆 #####################

# 保留第三方库android.support.v4不被混淆

-keep class android.support.v4.app.** { *; }

-keep interface android.support.v4.app.** { *; }

# 打包时忽略警告

-dontwarn android.support.**

# 如果你的项目中使用了第三方库,需要参考官方文档的说明来进行混淆配置
```

```
# 例如: 百度地图的配置 参考:http://developer.baidu.com/map/sdkandev-question.htm

#-keep class com.baidu.** { *; }

#-keep class vi.com.gdi.bgl.android.**{*;}

# 例如: 支付宝的混淆 参考: https://doc.open.alipay.com/doc2/detail.htm?treeId=59&articleId=103683&docType=1

#-libraryjars libs/alipaySDK-20150602.jar

#

#-keep class com.alipay.android.app.IAlixPay{*;}

#-keep class com.alipay.android.app.IAlixPay$Stub{*;}

#-keep class com.alipay.android.app.IRemoteServiceCallback{*;}

#-keep class com.alipay.android.app.IRemoteServiceCallback$Stub{*;}

#-keep class com.alipay.sdk.app.PayTask{ public *;}

#-keep class com.alipay.sdk.app.AuthTask{ public *;}
```

混淆规则配置文件配置好以后,需要在build.gradle中开启混淆

项目构建配置文件build.gradle

Project:build.gradle是用来配置项目的构建任务.默认的build.gradle内容如下:

```
//项目构建文件，你可以到各子项目/模块添加常用的配置选项.

buildscript {

//Android插件从这个仓库中下载

repositories {

jcenter() // 依赖仓库源的别名,兼容maven的远程中央仓库

}

//依赖

dependencies {

// android gradle插件

classpath 'com.android.tools.build:gradle:2.2.0-alpha1'

// 提示:

//请不要在此处添加应用程序依赖;它们应该在单个Module(模块)build.gradle文件中添加

//这里添加的应该只是Project的依赖

}

}

//此处配置Project中默认的仓库源,包括每个module的依赖

//这样每个module就不用单独配置仓库了

allprojects {

 repositories {

 jcenter()

 }

}

// 打包前执行clean任务
```

```
// 任务类型是 Delete

// clean任务就是删除项目根目录下的build目录(build为输出目录)

task clean(type: Delete) {

 delete rootProject.buildDir

}
```

buildscript中的repositories是指定Android插件的仓库源.

allprojects中的repositories是指定整个Project中默认的仓库源.

在Project Structure中设置

我们可以在Project Structure —> Project中设置Gradle和Android插件的版本,以及Android插件和默认第三方库的仓库源.

Project Structure和Project build.gradle对应的关系:

模块构建配置文件:build.gradle

Module:build.gradle是用来配置模块的构建任务.

默认的build.gradle文件内容如下:

```
//插件:

//这个module是一个android程序,使用com.android.application

//如果是android库,应该使用com.android.library

apply plugin: 'com.android.application'

android {//android程序构建需要配置的参数

//编译使用的SDK版本

compileSdkVersion 23

//buildtool版本

buildToolsVersion "23.0.2"

defaultConfig {//默认配置

applicationId "com.wirelessqa.basebuildsample" //apk包名

//最小SDK版本

minSdkVersion 16

//目标SDK版本

targetSdkVersion 23

//version code

versionCode 1

//应用程序的版本

versionName "1.0"

//android单元测试test runner

testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}
```

```
//构建类型, 此处配置debug和release版本的一些参数,像混淆、签名配置.

buildTypes {

//release版本的配置

release {

//是否开启混淆

minifyEnabled false

//指定混淆文件及混淆规则配置文件的位置

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

 }

 }

}

//模块依赖

dependencies {

//编译依赖libs目录下所有jar包

compile fileTree(dir: 'libs', include: ['*.jar'])

//编译依赖appcompat库

compile 'com.android.support:appcompat-v7:23.4.0'

}
```

模块构建配置文件:build.gradle也可以在项目结构中配置,下面我们详细介绍一下.

项目结构中配置模块构建

Project Structure用来配置项目和模块的各种构建参数和属性,前面我们已经介绍过了SDK Location和Project的配置,本章主要介绍模块构建的一些配置.

打开Project Structure对话框:

操作步骤：

菜单栏: File —> Project Structure

快捷键:

Mac: command + ;

Windows/Linux: Ctrl + Shift + Alt + S

工具栏:

Modules: 用来设置模块构建配置文件中的属性、签名、渠道特性、构建类型和依赖.

配置应用程序属性

Compile Sdk Version: 指定Android的编译版本.

对应build.gradle文件中的参数是:

```
compileSdkVersion 23
```

Build Tools Version: 指定构建工具的版本.

对应build.gradle文件中的参数是:

```
buildToolsVersion "23.0.2"
```

SDK编译版本和构建工具的版本都是我们已经下载到本地的,如果本地没有就不会出现在选择列表中.

Library Repository: 指定依赖的仓库源.

默认Project的build.gradle中已经指定了全局的仓库源,默认都是jcenter.因此这里可以为空. 如果重新指定,将会覆盖project build.gradle中的配置. 如果此处指定从maven中央仓库下载依赖,相应的module build.gradle文件中会新增下面的参数:

```
aaptOptions {

 ignoreAssetsPattern 'bixiaopeng'

}
```

Incremental Dex: 增量dex打包

开启此功能可以提升编译打包的速度,因为是增量打包而不是全量. 补充: Dex将.class文件转为Dalvik VM能识别的.dex文件 如果打开增量打包dex为true,相应的build.gradle文件中会增加下面的参数:

```
dexOptions {

 incremental true

}
```

配置应用程序签名

如果我们在这里配置了签名,相应的在module app的build.gradle文件中会自动添加下面的配置:

```
// 签名配置

signingConfigs {

MySigning {

 keyAlias 'myandroid'

 keyPassword '123456'

 storeFile file('/Users/quanke/myandroid.jks')

 storePassword '123456'

 }

}
```

对比一下使用maven打包,pom里的签名配置:

```
<!-- Sign apk -->

<!--签名文件路径-->

<sign.keystore>/Users/bixiaopeng/release.keystore</sign.keystore>

<!--签名文件别名-->

<sign.alias>laobi</sign.alias>

<!--签名文件别名密码-->

<sign.keypass>222222</sign.keypass>

<!--签名文件密码-->

<sign.storepass>222222</sign.storepass>
```

配置应用程序的特性

多渠道打包要在Flavors中配置相应的属性.

Name: Flavor的名字,如我们常用的渠道:xiaomi、baidu

Min Sdk Version: 向下兼容的最小SDK版本

Application Id: 应用程序的包名

Proguard File: 指定混淆文件路径,如果不指定会使用默认的.

Signing Config: 指定签名文件. 签名的文件在Signing中设置.

Target Sdk Version: 目标SDK版本

Test Instrumentation Runner: 指定Test Runner

Test Application Id: 测试应用的ID

Version Code: 应用程序的版本号,用于升级

Version Name: 应用程序的版本名称

Version Name Suffix：应用程序版本名称的后缀

新增一个Flavor:

新增一个Flavor,所有的参数都允许重新设置,如果不设置的都使用defaultConfig的配置.

```
// 产品特性

productFlavors {

 xiaomi {} //渠道名name为xiaomi

 baidu {

}

wandoujia {}

// 自动替换AndroidManifest.xml中的渠道号

productFlavors.all { flavor ->

 flavor.manifestPlaceholders = [CHANNEL_ID: name]

}

RC {

 minSdkVersion 20

 applicationId 'com.wirelessqa.basebuildsampleRC'

 proguardFile '/Users/bixiaopeng/myandroid.jks'

 signingConfig signingConfigs.MySigning

 targetSdkVersion 22

 testInstrumentationRunner 'android.support.test.runner.AndroidJUnitRunner'

 versionCode 1

 versionName '1.0.1'

 versionNameSuffix 'RC'

 }

}
```

针对不同的APP分发渠道,我们可以定义不同的productFlavors(产品特性).

也可以定义APP开发不同阶段的版本,比如:内测版本、灰度版本、正式版本.

也可以为不同的版本指定不同的ApplicationId(包名)，这样在同一个设备上可以同时安装两个版本,方便测试.

有了上面的配置你在执行打包命令的时候就会打出不同渠道的包.

配置应用程序的构建类型

构建类型 Build Types非常重要，在这里可以配置构建版本的一些非常重要的参数，默认有两个构建版本:debug和release。

默认的debug和release的区别只有Debugable(是否可调试),debug默认可调试,release默认不可以调试.

Name: 构建类型名称

Debugable: 是否可以调试

Jin Debuggable: Jin是否可以调试

Signing Config: 指定签名,同样是在signing中配置的,如果为空则不签名,打出来的包也是未签名过的.

Renderscript Debuggale: 是否使用渲染脚本(一种低级的高性能编程语言，用于3D渲染和处理密集型计算)

Renderscript Optim Level: Rendescript版本.

Minify Enabled: 是否混淆

Pseudo Locales Enabled: 是否支持本地化整理

Proguard File: 指定混淆文件路径

Application Id Suffix: 应用程序Id后缀

Version Name Suffix: 版本名称后缀

Zip Align Enabled: 是否支持Zip Align(字节码对齐)

新增一个构建类型:

修改后build.gradle文件中同步修改为:

```
buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile\('proguard-android.txt'\), 'proguard-rules.pro'

}

bbbbb {

debuggable true

jniDebuggable true

renderscriptDebuggable true

minifyEnabled true

pseudoLocalesEnabled true

proguardFile '\/Users\/bixiaopeng\/proguard-android.txt'

applicationIdSuffix '.daily'

versionNameSuffix '\_0705'

zipAlignEnabled true

}

}
```

如果我们指定打某个构建类型的包可以这样指定:

打出Debug的包: .\/gradlew assembleDebug

打出Release的包: .\/gradlew assembleRelease

打出Bbbbb的包: .\/gradlew assembleBbbbb

我们试着打出Bbbbb这种类型的包:

```
$ ./gradlew assembleBbbbb

:app:preBuild UP-TO-DATE

:app:preBaiduBbbbbBuild UP-TO-DATE

:app:checkBaiduBbbbbManifest


...

:app:packageXiaomiBbbbb

:app:assembleXiaomiBbbbb

:app:assembleBbbbb

BUILD SUCCESSFUL

Total time: 25.301 secs
```

在app\/build\/output\/apk目录下就会打出相应的apk包.

解析apk,查看包名和版本号跟我们配置的一致.

```
~$ aapt d badging /Volumes/FirstApp/app/build/outputs/apk/app-baidu-bbbbb-unsigned.apk

package: name='com.baidu.firstapp.daily' versionCode='1' versionName='1.0_0705'

sdkVersion:'8'

targetSdkVersion:'22'

....
```

> Project Structure中没有的配置但buildTypes中支持的属性还有很多,我们举个例子:

```


// 显示Log

buildConfigField "boolean", "LOG_DEBUG", "true"

// 移除无用的resource文件,minifyEnabled必须为true

shrinkResources true
```

配置应用程序的依赖

我们在这里添加模块中依赖的jar包、文件和模块,还可以配置它们的作用范围.

build.gradle中配置的依赖我们在上面已经介绍过,这里再提一下下,这两个依赖配置的意思是指定编译时需要依赖libs目录下所有的jar文件和一个android组件.

```
dependencies {

 compile fileTree(dir: 'libs', include: ['*.jar'])

 compile 'com.android.support:appcompat-v7:22.+'

 //此为新增依赖

 androidTestCompile 'junit:junit:4.12'

}
```

打签名包

操作步骤:

第1步: 签名证书配置窗口 : Build —> Generate Signed Apk —> 选择Module —> Next

然后弹出签名证书配置窗口:

我们可以点击 【Choose existing】来选择一个已存在的签名证书,也可以使用上次使用过的签名证书.

第2步: 输入选择证书的存储密码和密钥密码,如果不想每次都输入密码,请勾选 Remember passwords.

第3步: 点击Next,要求我们输入Master Password.

Master Password是用来保护我们存储在Android Studio数据库中的密码的.

如果忘记了密码是无法进行到下一步的.

怎么解决这个问题呢？

方法一: 在当前窗口中重置密码

点击 【Reset... 】—> 弹出重置Master密码窗口—> 输入新的密码 —> OK

点击 【Yes】 确认重置操作.

方法二: 到偏好设置中重置密码

如果你记得密码,直接输入密码.

Master password只会在我们第一次使用时要求输入

第4步: 配置应用程序构建类型和存储位置:

APK Destination Folder: 指定应用程序存储位置(默认放在app根目录下).

Build Type: 构建类型

Flavors: 渠道(没有就使用默认配置)

第5步: 点击[Finish]之后开始打包任务.

在Gradle Console工具窗口可以查看任务执行的过程:

打包成功后在IDE的右上角弹出打包成功提示.

生成的APK存储在第4步指定的位置, 默认在app\/build\/outputs\/apk目录下.

然后这个包你就可以发布到应用市场啦.

多渠道打包

国内Android应用下载有360、小米、豌豆荚、百度等等非常多的渠道，如果我们想统计每个渠道的下载量和活跃度,就需要使用统计平台.

我们以友盟统计为例,介绍如何配置渠道信息并执行自动化打包.

1.在AndroidMainfest.xml配置可动态替换的渠道参数

友盟集成文档中有说明,使用友盟统计需要在AndroidMainfest.xml配置相应的渠道号:

```
<meta-data

 android:name="UMENG_CHANNEL"

 android:value="xiaomi" /><!--渠道号为:小米-->
```

如果想动态的替换渠道号怎么办呢？

```
<meta-data

android:name="UMENG_CHANNEL"

android:value="${CHANNEL_ID}" /><!--动态替换渠道号-->
```

2.在build.gradle中配置渠道信息和自动替换脚本

```
// 多渠道打包

productFlavors {

 xiaomi {} //渠道名name为xiaomi

 baidu {}

 wandoujia {}


 // 自动替换AndroidManifest.xml中的渠道号

 productFlavors.all { flavor ->

 flavor.manifestPlaceholders = [CHANNEL_ID: name]

 }

}
```

配置好以后在Build Variants窗口中可以选择不同渠道的变种版本:

Gradle工具栏也会生成相应的任务:

3.默认配置

```
// 默认配置

defaultConfig {

 applicationId "com.wirelessqa.basebuildsample" //apk包名

 minSdkVersion 16

 targetSdkVersion 23

 versionCode 1

 versionName "1.0" //版本号

 //android单元测试test runner

 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}
```

所有渠道默认使用这一配置,如果渠道有特殊需求,可以在productFlavors对应的渠道号中单独配置.

4.打包后自动修改apk的名字

```
// 打包后自动修改apk的名字

// release包的命名格式为:产品名_版本号_渠道号.apk

// debug包的命名格式为:产品名_版本号_渠道号_Debug_打包时间.apk

applicationVariants.all { variant ->

 variant.outputs.each { output ->

 def outputFile = output.outputFile

 if (null != outputFile && outputFile.name.endsWith('.apk')) {

 File outputDir = new File(outputFile.parent);

 def baseName = PRODUCT_NAME + "${defaultConfig.versionName}" + "_" + variant.productF
lavors[0].name

 def newApkName



 if (variant.buildType.name.equals('release')) {

 newApkName = baseName + '.apk'

 } else if (variant.buildType.name.equals('debug')) {

 newApkName = baseName + "_Debug_${packageTime()}.apk"

 }



 output.outputFile = new File(outputDir, newApkName)

 }

 }



}
```

5.自动化打包

如何一次打出所有渠道的Release的包呢？

方法一: 命令行:

```
$ ./gradlew assembleRelease
```

方法二: Gradle工具窗口:

方法三: 菜单栏 —> Build —> Generate Signed APK —> 一步步下去 —> Flavors中全选—> Finish.

这样所有渠道的Release包都被打出来了.

6.查看渠道号是否被正确替换.

单击apk之后,Android Studio会自动解析apk,这样我们就可以在Android Studio中直接查看apk的信息了.

1.单击baidu这个渠道 —>2.点击AndroidManifest.xml —>3.查看渠道号为baidu

由此证明我们的打包脚本是OK的.

> 如果要打Debug包,执行assembleDebug任务就可以了.

如果只想打某一个渠道的包,执行对应的打包任务就可以了.

本例中全部的build.gradle脚本如下:

```
//插件:

//这个module是一个android程序,使用com.android.application

//如果是android库,应该使用com.android.library

apply plugin: 'com.android.application'



//产品名

def PRODUCT_NAME = "wirelessqa"

//打包时间

def packageTime() {

 return new Date().format("MMddhhmmss", TimeZone.getTimeZone("GMT+8"))
```

```
}

android {

// 签名配置
signingConfigs {
 MySigning {
 keyAlias 'myandroid'
 keyPassword '123456'
 storeFile file('/Users/bixiaopeng/myandroid.jks')
 storePassword '123456'
 }
}
// 编译sdk版本
compileSdkVersion 23
// 构建工具版本
buildToolsVersion "23.0.2"

// 默认配置
defaultConfig {
 applicationId "com.wirelessqa.basebuildsample" //apk包名
 minSdkVersion 16
 targetSdkVersion 23
 versionCode 1
 versionName "1.0" //版本号
 //android单元测试test runner
```

```
 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

 versionNameSuffix 'test'

}



// 产品特性

productFlavors {

 xiaomi {} //渠道名name为xiaomi

 baidu {

 }

 wandoujia {}

 // 自动替换AndroidManifest.xml中的渠道号

 productFlavors.all { flavor ->

 flavor.manifestPlaceholders = [CHANNEL_ID: name]

 }

}



// 构建类型,此处配置debug和release版本的一些参数,像混淆、签名配置.

buildTypes {

 // release包的配置

 release {

 //开启混淆

 minifyEnabled true

 // 指定混淆文件

 proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

 // 指定签名配置

 signingConfig signingConfigs.MySigning

 zipAlignEnabled true
```

```
 //移除无用的资源文件

 shrinkResources true

 }

}



// 打包后自动修改apk的名字

// release包的命名格式为:产品名_版本号_渠道号.apk

// debug包的命名格式为:产品名_版本号_渠道号_Debug_打包时间.apk

applicationVariants.all { variant ->

 variant.outputs.each { output ->

 def outputFile = output.outputFile

 if (null != outputFile && outputFile.name.endsWith('.apk')) {

 File outputDir = new File(outputFile.parent);

 def baseName = PRODUCT_NAME + "${defaultConfig.versionName}" + "_" + variant.productF
lavors[0].name

 def newApkName



 if (variant.buildType.name.equals('release')) {

 newApkName = baseName + '.apk'

 } else if (variant.buildType.name.equals('debug')) {

 newApkName = baseName + "_Debug_${packageTime()}.apk"

 }



 output.outputFile = new File(outputDir, newApkName)

 }

 }
```

```
 }

}



// 依赖的第三方库

dependencies {

 compile fileTree(include: ['*.jar'], dir: 'libs')

 compile 'com.android.support:appcompat-v7:23.4.0'

 compile 'com.android.support.constraint:constraint-layout:1.0.0-alpha1'

 compile 'com.android.support:design:23.4.0'

 testCompile 'junit:junit:4.12'

 androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'

 androidTestCompile 'com.android.support.test:runner:0.5'

 androidTestCompile 'com.android.support:support-annotations:23.4.0'

}
```

配置开发者服务

开发者服务中都是google提供的一些非常实用的服务,包括:广告、分析、身份验证、Firebase和通知.

第十二章 运行调试

调试是每个程序员必备的技能，代码总会出现问题，为了解决问题并验证程序的正确性，我们总会用到调试功能。Android Studio中强大的调试功能可以帮助我们快速定位问题。

运行和调试配置

打开运行和调试配置的方法：

菜单栏: Run —> Edit Configurations...

快捷键:

Mac: control + alt + R

Windows/Linux: Alt + Shift + F9 —> 0 工具栏: 点击Edit Configurations...

然后弹出配置界面:

Defaults列出了所有默认的配置, 点击 + 按钮可以新建一个新的Android启动/调试配置.

Android应用程序配置

Name:

名字. 可以在工具栏运行应用程序配置的下拉列表中看到。

General:

在这里配置安装、启动、部署应用程序选项

Module:

列表中列出了当前项目中的所有模块,我们可以指定相应的模块来运行.

Installation Options: 安装选项

1.Deploy:下拉列表中列出了应用程序运行时的部署模式

有三个选项:

Default APK: 部署默认的APK, 运行时会先打包安装,再启动APK。

Custom Artifact: 部署自定义的APK, 会根据你选择的模块来选择对应的配置。

Nothing: 不做任何部署,运行时会直接启动应用,如果应用已经安装了会直接启动, 没有安装就会报错。

2.Install Flags: 给**adb shell pm install** 添加运行参数**,**参数参加在**pm install**后面。

Install Flags为空时,运行应用程序时执行的命令是这样的:

```
# 1.把打好的包放到手机中的/data/local/tmp/目录下

$ adb push /Volumes/MyApplication/app/build/outputs/apk/app-debug.apk /data/local/tmp/
com.wirelessqa.myapplication

# 2.重新安装应用程序

$ adb shell pm install -r "/data/local/tmp/com.wirelessqa.myapplication"

pkg: /data/local/tmp/com.wirelessqa.myapplication

Success
```

如果添加一个参数-f

```
# 2.重新安装应用程序,在install后就多了个-f参数

$ adb shell pm install -f -r "/data/local/tmp/com.wirelessqa.myapplication"

pkg: /data/local/tmp/com.wirelessqa.myapplication

Success
```

Launch Option: 启动选项

1.Launch提供了四个选项.

Default Activity: 启动默认Activity,运行时会启动默认的MainActivity,如果没有会报错。

Specified Activity: 指定启动的Activity

在输入框中输入Activity的名字,输入时会有智能联想:

如果记不住名字,还可以搜索:

或在项目结构中查找:

定义好启动Activity后,运行应用时这个Activity就会被启动。

Nothing: 运行时不会启动任何Activity.

URL: 在这里可以指定启动的scheme.

2.Launch Flags: 给**adb shell am** 添加运行参数,参数添加在命令的最后面.

Deployment Target Options: 部署目标选项

Target:

Show Device Chooser Dialog：选择此选项,每次运行时都会弹出选择设备对话框。

USB Device: 使用USB连接的设备

Emulator: 使用模拟器.

Use same selection for future launches:

如果勾选此项,以后运行时都使用同样的选择,不需要再次选择了.

Miscellaneous:

在这里配置日志和安装选项

Logcat:

Show logcat automatically: 运行时自动显示logcat日志。

Clear log before launch: 启动前清空日志。

Installation Options:

Skip installation if APK has not changed： 如果代码没有变更,运行时跳过安装。

Force stop running application before launching activity。

启动Activity前强制关闭运行的应用程序.

Debugger:

在这里配置调试类型.

Debug类型包括: Java、Native、Hybrid.Profiling:在这里配置图形跟踪选项.

disable precompiled shaders and programs: 禁用预编译着色器和程序.

Before launch:

在这里可以配置运行之前需要执行的任务,默认会执行Make.

添加任务

点击+添加一个新的任务:

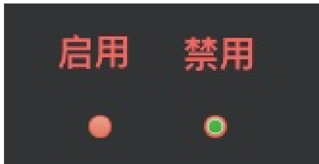
任务	说明
Run External tool	运行外部工具 ,如果有的话可以直接选择,如果没有可以去新建.
Make	编译选择的模块
Make Project	编译项目
Make, no error check	编译选择的模块，但不进行错误校验
Build Artifacts	构建构件, 如果有的话可以选择对应的构件.
Run Gradle task	运行Gradle任务
Gradle-aware Make	执行Gradlew任务

断点

断点会暂停应用程序的运行,线程被挂起. 然后我们可以通过调试器一行一行的查看代码.

断点的状态分为已启用断点和已禁用断点.

已启用的断点在调试状态下,应用程序每次执行到该断点时都会暂停.[图片]已禁用的断点在调试状态下,不会对应用程序的执行造成任何影响.



Android Studio中的断点分为很多类型, 每一种断点都有它适用的场合和特殊的作用,了解这些断点,才有可能更好的使用断点.

行断点

行断点是我们最常用到的断点, 它被用于对代码中特定的行进行调试.

操作步骤:

菜单栏: Run —> Toggle Line Breakpoint

快捷键:

Mac: command + F8

Windows\/Linux: Ctrl + F8

单击某行(非方法名所在行)左边栏即可设置断点

> 取消行断点跟添加行断点的方法同相。

属性设置

右击行断点弹出属性设置对话框:

Enabled: 断点启用和禁用.

Suspend: 勾选All,执行到断点时所有线程都会被挂起. 勾选Thread,执行到断点时只有当前断点所在的线程会被挂起.

切换线程挂起策略的时候,会显示Make Default,点击后就会把当前选中的策略变为默认的.

Condition:设置断点暂停条件.

方法断点

方法断点主要用来检查方法的输入和输出(参数和返回值).

操作步骤:

菜单栏: Run —> Toggle Method Breakpoint

单击方法名所在行的左边栏

属性设置

右击方法断点的图标,弹出属性设置对话框:

跟行断点相比,方法断点的属性中多了个Watch,用来设置是否监视方法的进入和退出.

Method entry 和 Method exit默认都被选中,也就是说调用此方法开始的时候和结束的时候断点都会被触发,如果不选中就不会触发.

字段观察点

当我们对程序运行的过程不太关心,只关心某个变量的变化的时候可以使用字段观察断点.

操作步骤:

菜单栏: Run —> Toggle Field Watchpoint

单击字段所在行的左边栏

属性设置:

右击变量的断点图标,显示属性设置:

这里有两个选择是Java Field Watchpoints特有的：

Field access: 当字段被访问的时候触发断点.

Field modification: 当字段被修改的时候触发断点.

条件断点

条件断点用来设置断点被触发的条件,如果条件不满足断点是不会被触发的.

操作步骤:

右击左边栏的断点图标 —> 勾选Condition —> 设置暂停条件

临时断点

当我们想某个断点只被触发一次后就自动删除的时候,可以使用临时断点.

操作步骤:

菜单栏: Run —> Toggle Temporary Line Breakpoint

快捷键：

Mac: fn + command + option + shift + F8

Windows\/Linux: Ctrl + Alt + Shift + F8

Alt + 鼠标单击左边栏

如果想把临时断点变为普通断点可以在属性中去掉Remove once hit勾选.

异常断点

异常断点会在某个异常发生时触发断点,这样我们就可以第一时间得到异常信息,方便排查问题.

操作步骤:

菜单栏: Run —> View Breakpoints

快捷键:

Mac: fn + shift + command + F8

Windows\/Linux: Ctrl + Alt + F8

然后在打开的断点窗口点击左上的角的+号按钮,选择Java Exception Breakpoints.

然后在弹出的[Enter Exception Class]窗口中输入你要调试的异常, 例如:NullPointException.

确定后当你的应用程序抛出空指针异常时,就会在异常处触发断点.

日志断点

在调试的时候,我们想临时多加一些日志但又不想重新构建应用程序的时候,就可以使用日志断点.

操作步骤:

右击断点 —> 取消勾选Suspend —> 在展开的选项中勾选Log evaluated expression —> 输入日志信息表达式.

因为取消了Suspend,所以执行到断点处不会暂停,而是会打印日志.

日志会打印在Console窗口中:

禁用断点

当某个断点我们暂时不需要,但又不想删除时可以先禁用.

操作步骤:

菜单栏: Run —> Toggle Breakpoint Enable

快捷键:

Mac: option +单击断点

Windows\/Linux: Alt +单击断点

禁用断点之后,在执行到该断点就不会暂停了. 如果想恢复断点使用禁用相同的操作即可.

断点设置

在断点的设置对话框中我们可以查看和管理项目中的断点,可以设置断点的属性.

打开属性配置窗口:

菜单栏 —> Run —> View Breakpoints

快捷键:

Mac: fn + shift + command + F8

Windows\/Linux: Ctrl + Alt + F8

调试工具窗口: 点击左边工具栏的View Breakpoints

右击断点的图标 —> 点击More

然后会打开断点设置对话框

调试工具

控制调试工具

控制调试工具用来管理调试当中的程序运行,提供了如下常用功能:

- 暂停、恢复程序运行;
- 终止进程
- 查看、禁用断点
- 获取线程堆栈

恢复程序运行

当程序在断点处暂停的时候,可以使用此功能来恢复程序运行. 如果有下一个断点, 就会跳转下一个断点处. 如果没有断点,程序就继续运行.

操作步骤:

调试工具栏: Resume Program

快捷键:

Mac: option + command + R

Windows\/Linux: F9

暂停程序运行

程序运行时点击此按钮来暂停程序运行.

操作步骤:

调试工具栏: Pause Program

终止进程

当我们想终止调试的时候可以直接终止进程.

操作步骤:

快捷键:

Mac: fn + command + F2

Windows\/Linux: Ctrl + F2

调试工具栏: Stop '配置名'

查看断点

操作步骤:

调试工具栏: View Breakpoints

快捷键:

Mac: fn + shift + command + F8

Windows\/Linux: Ctrl + Shift + F8

禁用断点

禁用断点功能可以切换断点状态(启用\/禁用). 我们可以暂时禁用项目中的断点去执行程序,这样就不会在断点处停止了.

操作步骤:

调试工具栏: Mute Breakpoints

获取线程堆栈

操作步骤:

调试工具栏: Get thread dump

恢复布局

恢复布局功能可以恢复到原始的布局,当前所有的布局变更都会被放弃.

操作步骤:

调试工具栏: Restore Layout

设置

操作步骤:

调试工具栏: Settings

Show Values Inline:

选中后启动内联调试功能,允许在编辑器中观察执行过的变量的值.

Show Method Return Values:

选中后会显示上次执行方法的返回值.

Auto-Variables Mode:

选中后调试器可以自动评估某些变量(在断点前几行和后几行的变量).

Sort values alphabetically:

选中后窗口中的变量值按字母的顺序排列.

Unmute breakpoints on session finish:

选中后当一个调试会话完成后,会重新启用所有禁用的断点.

步进调试工具

使用步进调试工具可以一行一行的查看代码的执行,就像程序执行的慢镜头回放一样,我们可以使用步进调试工具查看代码执行的过程.

显示执行点

当我们需要查看当前的执行点时,光标会立刻定位到当前执行到的断点.

操作步骤:

调试工具栏: 单击Show Execution Point

快捷键:

Mac: fn + option + F10

Windows\/Linux: Alt + F10

单步跳过

执行单步跳过,会执行下一行. 如果下一行是一个方法,不会进入方法体,而是会执行完此方法,然后跳到下一行.

操作步骤:

快捷键:

Mac: fn + F8

Windows\/Linux: F8

调试工具栏: 单击Step Over

单步进入

执行单步进入,会执行下一行. 如果下一行是一个方法, 且该方法如果是自定义的方法, 则会进入方法内. 如果不是自定义的(官方类库的方法)则不会进入.

操作步骤:

快捷键:

Mac: fn + F7

Windows\/Linux: F7

调试工具栏: 单击Step Into

强制进入

执行强制进入,会进入任何方法. 不管该方法是自定义的还是官方类库的.

操作步骤:

调试工具栏: 单击Force Step Into

快捷键:

Mac: fn + option + shift + F7

Windows\/Linux: Alt + Shift + F7

单步跳出

执行单步跳出,会跳出当前进入的方法,返回该方法被调用处的一下行.(注意:跳出意味着该方法被执行完毕)

操作步骤:

调试工具栏: 单击Force Step Out

快捷键:

Mac: fn + shift + F8

Windows\/Linux: Shift + F8

丢弃帧

如果你已经进入了某个方法内,执行丢弃帧,当前方法会被中断,并返回到当前方法被调用的地方. 另外变量的值也会回到最初.

当我们想反复调试某个方法,观察此方法一遍遍的执行的时候,可以使用此功能.

操作步骤:

调试工具栏: 单击Drop frame

运行到光标处

在调试时,光标可以放在任意一行. 然后执行Run to Cursor,程序会运行到光标所在行暂停. 其实这里的光标相当于一个临时断点.

操作步骤:

快捷键:

Mac: fn + option + F9

Windows\/Linux: Alt + F9

调试工具栏: 单击Run to Cursor

计算表达式

执行Evaluate Expression会打开计算表达式窗口.

操作步骤:

调试工具栏: 单击Evaluate Expression

快捷键:

Mac: fn + option + F8

Windows\/Linux: Alt + F8

计算表达式

当我们想临时修改某个变量的值或查看其内部方法返回值的时候,可以使用计算表达式功能.

Android Studio中提供了一个计算表达式和代码片段的功能, 使用起来非常方便.它除了支持正则表达式计算以外,还支持操作表达式,匿名表达式和内部类的计算.

两种计算模式:

Expression Mode: 计算单行表达式

Code Fragment Mode: 计算代码片段, 我们可以对声明、赋值、循环和if\/else进行计算.

使用表达式计算功能时需要注意:

1.只有在调试并且断点被触发的时候, 才可以使用计算表达式功能. 也就是说如果不是在调试的状态,计算表达式功能是不可用的.

2.如果计算表达式内调用一个方法,方法内又恰好有断点,那么该断点会被忽略,会直接计算出表达式的值.

在堆栈帧中计算表达式或代码片段

操作步骤:

第**1**步: 帧调试窗口中,选择你想要计算表达式的堆栈帧.

第**2**步: 调用计算表达式功能,有三种方法:

菜单栏 —> Run --> Evaluate Expression

快捷键:

Mac: fn + option + F8

Windows\/Linux: Alt + F8

在编辑器中光标定位处右击,选择Evaluate Expression

第**3**步: 选择计算模式

有两种计算模式,单行表达式和代码片段,可以通过中间的按钮[Code Fragment Mode]和[Expression Mode]来切换.

步骤**4**: 输入表达式或语句

基于你选择的模式输入表达式或语句,输入的时候会有补全提示:

> 如果你给对象设置了标签,那么在在输入表达式的时候可以通过标签来找到对应的表达式,Android Studio也会给出补全推荐:

步骤5: 执行计算

点击[Evaluate]来进行计算.

> 如果指定的表达式不能计算或是有错误,点击[Evaluate]后会给出可能的错误原因.

计算任意表达式

在变量调试窗口中右击变量 —> Evaluate Expression

会进入表达式计算窗口

之前选中的变量会显示在表达式输入框中.

可以切换计算模式、查看计算过的历史、输入表达式

快速计算表达式的值

前提条件:

光标放在放达式上或选中表达式

操作步骤：

菜单栏: Run --> Quick Evaluate Expression

快捷键:

Mac: fn + option + command + F8

Windows\/Linux: Ctrl + Alt + F8

选中表达式立即显示表达式的值

在调试工具栏上点击设置按钮,勾选[show value on selection change]

然后再去选中表达式,会在表达式的上面显示计算出来的结果.

> 此功能默认是不开启的

关联调试到Android进程

通常我们调试应用程序时,需要先添加断点,再运行调试(Debug). 这样做会比较慢,因为需要重新部署(打包、安装、运行)应用程序.

Android Studio提供了一种方法可以随时调试应用程序,不管当前应用程序是否以调试模式运行.

当我们想快速调试一个正在运行的应用程序时,可以使用此功能.

前提条件:

应用程序已经在设备上运行,已添加断点.

操作步骤:

菜单栏: Run —> Attch Debugger to Android Process

然后弹出进程选择对话框,选择需要调试的应用程序的进程.

确定后,调试器窗口被激活,接下来就可以正常的调试应用程序啦.

配置和运行单元测试

配置和运行本地单元测试

本地单元测试

本地单元测试用来执行那些对Android没有依赖或Android依赖容易mock的单元测试.

本地单元测试运行在自己电脑上,测试用例在本地虚拟机上编译运行,执行速度快.

本地单元测试写在app\/src\/test\/java目录下.

本地单元测试使用JUnit或TestNG测试框架.

使用的演示代码

这里使用Google官方开源的示例进行演示,地址:https:\/\/github.com\/googlesamples\/android-testing\/tree\/master\/unit\/BasicSample

使用的测试框架

我们用JUnit测试框架举例.

使用IDE运行本地单元测试

打开BasicSample项目,在EmailValidatorTest编辑界面或项目窗口中的类名上右击 —> Run 'EmailValidatorTest'

这样EmailValidatorTest这个测试类里面的所有测试用例都会被执行.

> 如果只想执行某一个测试用例,在测试用例名上右击,选择‘Run 用例名’来执行.
>
> 如果想某个目录下所有的用例都被执行,在目录名上右击,选择‘Run Test in 目录名’来执行.

保存临时配置

通过上面的方法运行完测试之后,在工具栏的运行配置列表中显示了刚才运行的测试类. 这个配置是自动生成的,如果需要保存,点击 [Save '测试类名' Configuration].

配置本地单元测试

如果我们需要自定义本地单元测试配置,可以这样做.

操作步骤:

打开Run\/Debug Configurations对话框 —> 点击左上角的+号 —> JUnit

然后新增一个JUnit配置窗口:

1.Name: 输入配置名

2.Use classpath of module: 选择测试模块,下拉列表中会列出项目所有的模块,选择我们需要测试的.

3.Test kind: 选择一种测试范围,本例中选择的是Class

4.测试用例配置: Class: 选择测试类

测试用例配置项跟我们选择的测试范围是相关联的.

如果Test kind选择Class,显示测试类的配置项.

如果Test kind选择Method,显示测试类和方法的配置项.

其它Test kind与此类似,这里就不一一列举了.

5.Fork mode: 用来指定是为每个测试用例都创建一个进程,还是所有测试用例在一个进程中执行.

两个选项:

none: 在一个进程中执行所有测试[图片]执行结果: 耗时8ms

method：每个测试用例都会创建一个测试进程,这种方式会比较慢.[图片]执行效果: 耗时19ms

6.Repeat:

Once：所有用例只执行一次[图片]N Times: 自定义用例执行次数,比如:设置5次,每一条用例都会被执行5次.[图片]Until Failure: 不断重复执行,直到用例失败.[图片]Until Stopped: 不断重复执行,直到手动停止.

7.VM options: 默认值为-ea,用来设置jvm是否启动断言机制.

8.JER环境,使用默认的就好了.

使用命令行运行单元测试

```
./gradlew test
```

执行gradle测试任务后所有的测试用例都会被运行,然后会产出测试报告.

测试报告存放在址: app\/build\/reports\/debug\/index.html

通过gradle工具栏执行的效果同命令行.

配置Android单元测试

如果我们需要自定义本地单元测试配置,可以这样做.

操作步骤:

打开Run\/Debug Configurations对话框 —> 点击左上角的+号 —> Android Tests

然后新增一个Android Tests配置窗口:

1.Name: 输入配置名

2.Module：选择需要测试的模块

3.Test: 指定测试范围

4.指定instrumentation runner

5.指定目标设备.

> 运行Android单元测试方法同运行本地单元测试.

第十三章 工具

第十四章 版本控制

目前支持CVS、SVN、Git、Mercurial等主流的版本控制系统。在**Android Studio**中使用版本控制系统要先确保你电脑上已经配置好了相应的工作环境。

版本控制系统

假设我们打开的是一个没有使用版本控制的项目，首先启用Git作为项目的版本控制系统:

菜单栏: VCS —> Enable Version Control Integration —> 选择Git。

然后项目根目录下就多了个.git，项目中的文件都是红包的，表示未加入Git版本跟踪。

开启版本控制集成后，我们就可以看到Android Studio中的版本控制系统都有哪些操作入口了。

一. 菜单栏->VCS

查看菜单栏上的VCS，可以看到版本控制系统的所有功能

二. 右键 ->菜单

右击要操作的文件或文件夹，在弹出的菜单中可以看到版本控制系统的选项，本例是Git，如果你使用的是svn，这里会显示Subversion。

右键菜单中提供的功能，只包含对应的版本控制系统特有的功能，没有包含通用的功能，这是菜单栏的区别。

这里提供的Git操作区分了本地和远程的操作，操作分类更加明确。

三. 底部工具栏 -> Version Control

底部工具栏会显示Version Control，如果没有显示，请点击View —> Tool Windows来显示。

打开底部工具栏上的Version Control会打开操作面板

这里很多功能跟菜单栏的VCS中是相同的，但是这里操作更加有针对性，还可以批量操作，具体用法我们后面讲。

四. 顶部工具栏 ->常用功能

顶部工具栏显示了我们版本控制当中最常用的几个功能

从左到右依次是:更新项目、提交变更、跟远程仓库中的文件进行对比、显示历史、撤消操作。

五. 状态栏 -> 分支操作

状态栏提供了Git的分支操作，在这里我们可以对本地或远程的分支进行各种操作，像检出、创建新的分支、对比、合并、删除等。

这些功能具体怎么用?

且听下回分解

Git偏好设置

当我们把Git的环境配好，在Android Studio的偏好设置中只需要使用默认的配置就可以了。一般不需要特殊配置，但也不排除你有特殊的需求，那下面我们就介绍下Git的偏好设置。

设置步骤:

偏好设置: Version Control —> Git

Path to Git executable:

Git执行路径,这里使用的是默认路径,如果你自定义了Git路径,这里要记得改一下, 不然会报错的.

SSH executable:

执行SSH,默认是使用Android Studio内建的,你还可以选择使用Native(本地)的.

commit automatically on cherry-pick:

在cherry-pick时是否自动提交.

tips：cherry-pick可以选择某一个分支中的一个或几个commit来进行操作.

warn if crlf line separators are about to be committed:

如果回车换行符被提交是否警告

Warn when committing in detached HEAD or during rebase:

提交detached HEAD或rebase时是否警告.

tips： detached HEAD用来让HEAD随便指向某个commit id，rebase用来合并代码.

Update method:

选择代码更新方式:默认分支\/merge\/rebase

Auto-update if push of the current branch was rejected:

如果push当前分析被拒绝是否自动更新.

Allow force push:

是否允许强制push.

配置Github

为了后面方便的使用Github来管理代码，我们需要配置好Github的账户信息。

操作步骤:

第1步: 进入偏好设置

偏好设置：Version Control —> GitHub

第2步: 输入账户信息

1.输入Host: https:\/\/github.com

2.在Login输入用户名，在Password输入密码 (如果没有请先去注册)

3.Test连接是否成功

从Github克隆代码

第一步: 进入Github克隆界面

如果我们想使用Github上的开源项目，可以使用Android Studio直接下载项目代码。

操作步骤:

第1步: 进入Github克隆界面

三个路径:

欢迎界面: Check out project from Version Control —> Github

菜单栏: File —> New —> Project from Version Control —> Github

菜单栏: VCS —> Checkout from Version Control —>Github

第二步:配置克隆项目地址

配置Github repository

> 点击【 Test 】可测试能否连接成功。

第三步:下载并打开项目

点击【 Clone 】开始从给定的仓库地址克隆项目到本地，完成后会弹出一个【 Import Project from Gradle 】确认对话框。

点击【 OK 】打开项目。

将本地项目共享到GitHub

如果我们本地的项目想放到GitHub上开源，可以直接将本地项目共享到GitHub。

操作步骤:

第1步: 点击【 Share Project on GitHub 】

打开本地项目—> 菜单栏 —> Import into Version Control —> Share Project on GitHub。

第2步: 确定是本地项目

如果此项目已经在Github上，在Android Studio右上角会有相应的提示信息，点击Github可以跳转到项目主页。

如果此项目没有在Github上，会弹出一个新建仓库的对话框:

第3步:指定首次提交的文件和提交的备注

点击OK以后在状态栏会显示进度.

第4步:确认分享成功

分享成功后在Android Studio的状态栏和右上角会显示成功提示.

如果想查看Github上到底有没有分享成功, 后点击[SecondApp]到项目主页查看.

查看本地变更历史

当你想查看某个文件或文件夹本地变更的历史记录的时候,可以使用此功能.

一. 查看某个文件的本地变更历史

前题条件:

在文件列表中选中此文件或打开此文件,光标在文件中.

操作步骤:

方法一: 点击右键 —> 在弹出的操作选项中点击Local History —> Show History

方法二: 菜单栏 —> VCS —> Local History —> Show History

显示变更历史

如果两个文件有不同会提示不同的个数,窗口底部列出了不同颜色表示的变更操作.

二. 查看某段代码的本地变更历史

如果你想查看某段代码本地的变更，需要先选中这段代码,然后在Local History列出的选项中选择[Show History for Selection]

三. 查看某个文件夹的本地变更历史

如果想要查看某个文件夹的本地变更只需要选中这个文件夹,然后右击选择 Local History — > Show History就可以了.

我们还可以为每次变更贴上一个标签(Put Label),只要在Local History中选择Put Label，然后输入标签内容就可以了.

Git添加文件

在讲添加文件之前我们先要了解下git中文件的状态和提交流程.

我们的工具目录下的文件只有两种状态: 未跟踪和已跟踪.

未跟踪是指没有被纳入版本控制的文件,git就不会跟踪文件的变更.

已跟踪是指已被纳入版本控制的文件,在上一次的快照中有它们的记录,在工作一段时间后,这些文件的状态可能是未修改、已修改或已放入暂存区.

本地文件提交到远程仓库一般的流程为:

添加文件跟踪 —> 暂存已修改的文件 —> 提交到本地仓库 —>推送到远程仓库.

本节我们讲添加文件跟踪和暂存已修改的文件.

一. 添加文件跟踪

如果我们新建了一个文件,默认是没有被跟踪的,git就不会跟踪文件的变更,那如何跟踪文件呢?

举个例子:

在Android Studio中新建一个文件Demo.java,然后就会弹出一个对话框,来询问你是否要将此文件添加到Git.

点击[Yes]此文件就被跟踪了.

上面的这个方法很方便,在你新建一个文件的时候就提示你进行文件跟踪,但如果你用别的方式新建文件,那怎么样添加跟踪呢?

举个例子:

我们通过命令新建一个文件Demo.java,请注意,文件不同的状态都有不同的颜色来表示.

1.未跟踪的文件显示为红色

那我们如何将Demo添加到跟踪呢?

右击Demo文件或在Demo文件编辑区右击 —> Git —> Add

或者点击菜单栏 —> VCS—> Git —> Add

添加完成后,Demo文件颜色变为了绿色,所以

2.已跟踪的文件显示为绿色

3.提交到本地仓库\/远程仓库的文件颜色显示为白色

4.已修改的文件显示为蓝色

二. 暂存已修改的文件

暂存已修改的文件方法同添加文件跟踪.

所以add功能可以用来跟踪新文件，或者把已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等,我们可以将这个功能理解为“添加内容到下一次提交中”.

不过如果想提交起来更简单,可以直接使用commit命令,在提交配置列表里可以选择已修改的文件和新添加的文件,在commit之前会自动add选中的文件。

Git提交变更

当本地文件变更以后,可以通过VCS —> Git —> Commit File 弹出提交变更窗口.

当然,分支合并过后也会弹出提交变更窗口.

配置提交信息

提交变更窗口中你可以选择Change list,也可以选择要提交的变更文件,默认是全选的.

在Author中选择或者输入作者名字.选择Amend commit(修订提交)会在Commit Message中添加上一次的提交信息.

在提交之前,你还可以选择做一些代码优化的工作,比如: Reformat code(格式化代码)、Rearrage code(重新排列代码)、Optimize imports(优化导入)、Perform code analysis(执行代码分析)、Check TODO(检查待办事项)、Cleanup(清理)、Update copyright(更新版权声明).

点击某个变更的文件, 在Details中会显示本地和远程的对比.

提交变更

当配置好提交信息以后,将鼠标放到Commit上面,会弹出提交操作列表.

Commit and Push: 将本地变更的文件提交到本地仓库,然后推到远程仓库.Create Patch: 将本地变更的文件作为补丁创建.Commit: 将本地变更的文件提交到本地仓库.

这里我选择Commit and Push：

如果你选择了提交之前进行一些代码检查或优化,会先执行优化.

如果点击Review可以查看代码分析出来的问题,查看以后觉得没问题,重复上面的操作(Android Studio会有记录)

如果点击Commit会直接提交.

Commit成功后会提示Push到远程仓库.

默认提交到当前分支，你也可以选择其它分支.

点击Push将本地变更推送到选择的远程分支.

Git文件逐行追溯

如果你想查看某个文件的某一行是谁修改的,可以使用文件逐行追溯功能.

执行文件逐行追溯操作,会显示某个文件每一行的详细改动信息，也可以说是每一行的注释, 注释包括修订版本号、提交者、提交日期以及提交次数.

如何进行文件逐行追溯？

其实就是执行git annotate命令,同git blame.

在Android Studio中执行annotate的操作路径有下面这几个:

方法一: 右击文件左边状态栏 —> 选择Annotate

方法二: 右文件编缉区域 —> 点击 Git —> 选择Annotate

方法三: 菜单栏 —> 点击VCS —> 选择Annotate

选择Annotate后会显示文件每一行的注释,注释如下:

文件中每一行的注释包括修订版本号、提交者、提交日期以及提交次数.

如果你想追溯某一行的改动,点击这一行的注释,会弹出那次提交的改动文件列表以及提交信息.

因此如果你想在进行逐行追溯的时候省点力气,或者让review代码的人轻松一点 ,那就要注意,每次提次不要改动太多文件,提交信息(commit message)一定要写清楚改动的目的以及影响点,这些可以写到团队协作规范里.

显示当前修订版本

修订版本号(revision)是一组SHA-1值,我们可以通过SHA-1值来获取对应的那一次提交.

> tips: SHA-1是散列函数加密算法, 输出的散列值为40位十六进制数字串,可用于验证信息的一致性,防止被篡改.

android studio中显示当前修订版本的操作步骤是:

右击某一个文件或在文件编辑区右击 —> Git —> Show Current Revision

或者点击菜单栏 —> VCS—> Git —> Show Current Revision

Git文件比较

Android Studio集成的Git提供了丰富的文件比较功能,我们可以将本地文件与远程仓库中的、某次提交的或其它分支的文件进行比较.

可以通过如下操作方法使用比较功能:

方法一: 右击某一个文件或右击文件的编辑区 —> Git.

方法二: 菜单栏 —> VCS —> Git

方法三: Version Control —> 右击有变更的文件 —> Git

比较功能有下面这几个:

Compare with the Same Repository Version:比校本地文件与远程仓库的文件

Compare with Latest Repository Version:本地文件与最近的一次提交比较

Compare with:本地文件与某一次提交比较

Compare with Branch:本地的文件或文件夹与某个分支上的进行比较.

接下来我们来介绍一下这几个功能.

比校本地文件与远程仓库的文件

如果本地某一个文件被修改了,我们想查看它与远程仓库中的文件有什么不同,就需要比较一下.

比较方法:

右击某一个文件或右击文件的编辑区 —> Git —> Compare with the Same Repository Version

本地文件与提交的版本进行比较

Compare with Latest Repository Version: 本地文件与最近的一次提交比较

Compare with: 本地文件与某一次提交比较,点击此选项会显示文件的提交列表,可选择某一次提交进行比较.

前面讲的几个比较功能都是同一个分支上的文件的比较,那如果想跟其它分支上的文件或文件夹进行比较应该怎么办呢?

比较不同分支的文件或文件夹

Compare with Branch: 本地文件或文件夹与某个分支上的进行比较.

Git撤销操作

如果你对某个或某几个文件进行了修改,现在想撤销这些修改,应该怎么办呢?

请使用Git提供的撤销操作功能.

如下图所示,Android Studio中提供了多个快捷操作方式,可以方便的撤销操作.

确认Revert后,被选中的文件就恢复到了改动之前的状态.

Git版本回退

Git版本回退的意思就是将本地代码回退到某一个指定的版本,此版本之前的所有内容都会被重置.

操作方法:

VCS —> Git —> Reset HEAD—> 弹出Reset Head对话框.

Reset HEAD：

Reset Type: 回退类型

Mixed：回退到某个版本,本地源码不会回退, 会回退commit和index信息.

Soft：回退到某个版本, 本地源码和index信息不会回退, 只回退了commit的信息,如果还要提交，直接commit即可.

Hard：彻底回退到某个版本，本地的源码也会变为某个版本的内容.

To Commit: 回退版本

在To Commit中配置要回退到哪个版本,默认是HEAD.

HEAD: 回退到最近一个提交版本.

HEAD^: 回退到上一个提交版本.

HEAD^^: 回退到上上一个提交版本.

… 依此类推.

也可以使用另外一种方式回退版本:

HEAD~0:回退到最近一个提交版本.

HEAD~1:回退到最近一个提交版本.

HEAD~2:回退到最近一个提交版本.

… 依此类推.

Validate: 版本验证

在指定回退到某个版本之前你也可以先验证一下是否是你想回退的版本.

确认后点[Reset]就可以回退到指定的版本了.

Git查看提交历史

我们在13.6节介绍过了【查看本地变更历史】,在这一节我们介绍Git查看提交历史记录功能.

跟本地变更历史提供的功能相同,Git查看提交历史记录也都有查看文件、文件夹和代码段的历史记录功能，不同点在于【查看本地变更历史】仅能查看本地的变更,远程的改动是无法看到的,【Git查看历史记录】可以查看所有commit以后的历史.

因此当你想查看某个文件或文件夹提交的历史记录的时候,可以使用此功能.

一. 查看某个文件的提交历史

前题条件:

在文件列表中选中此文件或打开此文件,光标在文件中.

操作步骤:

点击右键 —> 在弹出的操作选项中点击Git —> Show History —> Show History

然后会在Version Control中显示History

二. 查看某段代码的提交历史

如果你想查看某段代码提交历史记录，需要先选中这段代码,然后在Git操作列表中选择[Show History for Selection]

Git分支管理

什么是分支？

当我们在进行软件开发时, 同一个软件多个人协同开发,因此要有不同的分工,如果想让彼此的代码不受影响,那就需要在不同的分支上进行开发,开发完成后再进地合并.

分支可以理解为一个主干衍生出来的支干,我们可以在这些支干上修改代码,且彼此不受影响,这样做的好处就是在同一个数据库里可以同时进行多个修改,最终会合并到一起.

Android Studio中Git的分支管理特别方便,我们可以通过Git操作列表中的branchs或状态栏的分支管理来对本地或远程的分支进行各种操作,像检出、创建新的分支、对比、合并、删除等.

如上图,Local Branches列出了本地所有的分支,这些分支不一定全部都push到了远程仓库,Remote Branches列出了所有的远程仓库中的分支.

新建分支

点击[New Branch] —>输入分支的名字 —> OK

检出分支

检出分支作为新的本地分支,意思就是你可以把一个远程分支checkout下来,可以起不同的名字,可以是多个本地分支,但是提交时都是向同一个远程分支进行提交.

操作步骤:

选择一个分支(远程或本地分支) —> 点击 [Checkout as new local branch]

如果是远程分支会默认使用远程分支名作为本地分支名.

但如果本地已经有了这个分支名,会提示你重命名.

输入新的分支名 —> OK —> 成功检出.

如果你是对本地分支进行 [Checkout as new local branch]操作,会直接弹出一个[Checkout New Branch]输入框,不会有默认的分支名.

检出分支V标签V修订版本

如果我们想检出某个分支\/标签\/修订版本,只需要知道对应的名字就可以.

在Git Branches操作列表中选择[Checkout Tag or Revision] —>在弹出的输入框中输入分支名 —> OK后会检出应的分支.

检出tag也只需要输入tag名就可以了,如果检出当前分支的某个修订版本需要输入对应的修订版本号.

切换分支

切换分支需要选择要切换的分支,点击[Check out]

如果你原来的分支没有变更,checkout后会切换到新的分支,如果有变更且有冲突,会先让你解决冲突然后再切换.

分支对比

分支对比是拿当前分支与另外一个分支(本地或远程)进行对比

对比两个分支的提交日志

对比两个分支的不同

合并分支

合并分支只将选中的分支与当前的分支进行合并

如果有冲突会弹出冲突列表,解决冲突后可合并成功,合并冲突请参考13.17节.

删除分支

删除本地分支:本地分支会被直接删除

删除远程分支: 删除远程分支会有确认是否删除,确认删除后状态栏会有正在删除的信息显示,删除成功后Version Control上会有删除成功提示,Remote Branches列表中此分支也会消失.

提交分支

提交分支就是将本地Push到远程仓库

Git创建标签

Git 可以给历史中的某一个修订版本打上标签,通常我们会使用标签来表示一个版本的发布.

操作步骤:

VCS —> Git —>Tag —>弹出创建标签配置窗口

配置介绍:

Git Root: 项目地址

Current Branch: 显示当前的分支

Tag Name: 标签名

Force:如果相同的tag名已经存在了,是否强制创建.

Commit: 通过输入的修订版本号、tag名来打标签,如果空白默认对HEAD打标签.

Create Tag:

另外我们还可以在Version Control中,右击某个提交信息然后创建标签.

本地创建标签成功以后,push到远程仓库就可以了.

第十五章 偏好设置

外观

设置Android Studio的主题

设置Android Studio的字体和大小

设置工具提示的延迟时间

在状态栏显示内存状态

调整演示模式的字体大小

菜单和工具栏

对菜单选项和工具栏的工具进行增删改

设置启动Android Studio时是否自动打开项目

设置退出Android Studio时是否弹出确认提示

设置打开一个项目时的打开方式

设置文件自动保存的时间

设置密码存储策略

设置HTTP代理

设置Android Studio版本更新规则

设置是否让Google统计你的使用信息

管理Android SDK平台

使用单独的SDK管理窗口来管理SDK

管理Android SDK开发工具

管理Android SDK更新站点

文件颜色

范围

通知

快速列表

路径变量

Android Studio中使用Eclipse快捷键

自定义keymap

第三部分 编辑器

常规设置

编辑器常规设置

设置点击编辑器光标定位在一行的结尾或定位在点击的位置

设置鼠标悬停在元素上会显示文档提示

设置是否自动换行

设置粘贴历史记录的个数

置记录最近打开项目的个数

设置自动导入

设置粘帖时自动导入包

设置自动导入需要的包

设置是否弹出导入提示

设置外观

设置编辑器一直显示行号

设置编辑器显示方法分隔符

设置编辑器显示空格

设置编辑器显示缩进向导

设置代码补全

设置自动补全时是否区分大小写

加快自动弹出代码补全提示的速度

关闭自动弹出代码补全提示

设置查看方法参数信息的时候显示方法签名

设置代码折叠

设置控制台

设置编辑器标签

设置用星号标记修改过的文件标签

设置打开的文件标签可以多行显示

设置文件标签的显示位置

设置打开的文件标签超过一定数量时的关闭规则

设置后缀补全

设置智能键

开启使用驼峰单词

颜色和字体

切换编辑器配色方案

为编辑器添加自定义配色方案

设置代码的字体和大小

设置是否显示条标和条标的显示颜色

设置控制台的颜色

第四部分 插件

第五部分 版本控制

第六部分 构建,执行,部署

构建工具

设置开启Gradle离线工作模式

编译设置自动编译项目

设置并行编译

调整编译内存大小

第七部分 语言与框架

第八部分 工具

第十六章 插件

Google 在2013年5月的I/O开发者大会推出了基于IntelliJ IDEA Java IDE上的Android Studio。AndroidStudio是一个功能齐全的开发工具，还提供了第三方插件的支持。让开发人员更快速更好的开发程序。

插件下载安装

- in Android Studio: go to `File → Settings → Plugins → Browse repositories` and search for `插件名`

或者

- in Android Studio: go to download it jar and install `File → Settings → Plugins → Install plugin from disk`

常用插件

数据来源：https://github.com/dreamlivemeng/androidstudio-plugins

1. Android ButterKnife Zelezny ButterKnife是一个专注于Android系统的View注入框架,可以减少大量的findViewById以及setOnClickListener代码，可视化一键生成。####PS:效果图就不贴了，打开插件下载地址和源码地址都能看见，而且数据多了加载效果图蛮卡的。 插件下载地址：https://plugins.jetbrains.com/plugin/7369?pr=androidstudio 插件源码地址： https://github.com/avast/android-butterknife-zelezny 插件教程：http://blog.csdn.net/dreamlivemeng/article/details/51261170 推荐指数：五星

2. GsonFormat GsonFormat是一个快速格式化json数据,自动生成实体类参数的插件。 插件下载地址：https://plugins.jetbrains.com/plugin/7654?pr=androidstudio 插件源码地址：https://github.com/zzz40500/GsonFormat 插件教程：http://blog.csdn.net/dreamlivemeng/article/details/51262538 推荐指数：四星

3. Android Drawable Importer 为了适应所有Android屏幕的大小和密度，每个Android项目都会包含drawable文件夹。任何具备Android开发经验的开发人员都知道，为了支持所有的屏幕尺寸，你必须给每个屏幕类型导入不同的画板。Android Drawable Importer插件能让这项工作变得更容易。它可以减少导入缩放图像到Android项目所需的工作量。Android Drawable Importer添加了一个在不同分辨率导入画板或缩放指定图像到定义分辨率的选项。这个插件加速了开发人员的画板工作。 插件下载地址：https://plugins.jetbrains.com/plugin/7658?pr=androidstudio 插件源码地址：https://github.com/winterDroid/android-drawable-importer-intellij-plugin 插件教程地址：http://blog.csdn.net/lee_sire/article/details/49684385 推荐指数：三星

4. android-selector-chapek / SelectorChapek for Android 根据资源自动生成相应的selector。 插件下载地址：https://plugins.jetbrains.com/plugin/7298 插件源码地址：https://github.com/inmite/android-selector-chapek 推荐指数：四星

5. Android Parcelable code generator 快速实现Parcelable接口的插件。 插件下载地址：https://plugins.jetbrains.com/plugin/7332?pr= 插件源码地址：https://github.com/mcharmas/android-parcelable-intellij-plugin/ 插件教程地址：http://blog.csdn.net/kroclin/article/details/40902721 推荐指数：四星

6. Markdown supportMarkdown 是一种可以使用普通文本编辑器编写的标记语言，通过类似HTML的标记语法，它可以使普通文本内容具有一定的格式。 插件下载地址：https://plugins.jetbrains.com/plugin/7793 插件文档地址：https://github.com/JetBrains/intellij-plugins/tree/master/markdown 推荐指数：四星

7. Markdown Navigator一款Markdown插件，是https://github.com/nicoulaj/idea-markdown他的一个分支，但是主项目由于维护的原因已经从jetbrains中删除了，如果想了解主项目的也可以通过上面的github地址进行了解。 插件下载地址：https://plugins.jetbrains.com/plugin/7896?pr= 插件源码地址：https://github.com/vsch/idea-multimarkdown 推荐指数：四星
8. Android Postfix completion 可根据后缀快速完成代码。 插件下载地址：https://plugins.jetbrains.com/plugin/7775?pr= 插件教程地址：http://blog.jetbrains.com/idea/2014/03/postfix-completion/ 推荐指数：五星
9. AndroidAccessors 快速实现get和set方法的插件。 插件下载地址：https://plugins.jetbrains.com/plugin/7496?pr= 插件文档地址：https://github.com/jonstaff/AndroidAccessors 推荐指数：三星
10. Lifecycle Sorter 可以根据Activity或者fragment的生命周期对其生命周期方法位置进行先后排序。 插件下载地址：https://plugins.jetbrains.com/plugin/7742?pr= 插件源码地址：https://github.com/armandAkop/Lifecycle-Sorter 推荐指数：五星
11. ADB WIFI无需root就能wifi调试。 插件下载地址：https://plugins.jetbrains.com/plugin/7856?pr= 插件源码地址：https://github.com/layerlre/ADBWIFI 推荐指数： 五星
12. ADB Idea adb 调试工具,Uninstall App、Kill App、Start App、Restart App、Clear App Data、Clear App Data and Restart插件下载地址：https://plugins.jetbrains.com/plugin/7380?pr= 插件源码地址：https://github.com/pbreault/adb-idea/ 推荐指数：五星
13. Android WiFiADB 无线调试应用 插件下载地址：https://plugins.jetbrains.com/plugin/7983 插件源码地址：https://github.com/pedrovgs/AndroidWiFiADB 推荐指数：五星
14. CodeGlance 最大的用途：可用于快速定位代码。 插件下载地址：https://plugins.jetbrains.com/plugin/7275?pr= 插件源码地址址：https://github.com/Vektah/CodeGlance 推荐指数：五星
15. JSONOnlineViewer 可实现直接在android studio中调试接口数据，可以选择请求类型，自定义请求头及请求体，json数据格式化后展示 插件下载地址：https://plugins.jetbrains.com/plugin/7838?pr= 推荐指数：四星
16. FindBugs-IDEA 通过FindBugs帮你找到隐藏的bug及不好的做法。 插件下载地址：https://plugins.jetbrains.com/plugin/3847?pr= 插件源码地址：https://github.com/andrepdo/findbugs-idea/tree/master 插件教程地址：http://blog.csdn.net/fancy_xty/article/details/51718687 推荐指数：四星

17. jimu Mirror 这是一个可以让你在真实的设备上迅速测试布局的插件。jimu Mirror允许在设备上预览随同编码更新的Android布局。插件下载地址：https://plugins.jetbrains.com/plugin/7517?pr= 插件教程地址：http://www.itnose.net/detail/6204426.html 推荐指数：四星

18. JavaDoc 添加注释，可自定义模板。 插件下载地址：https://plugins.jetbrains.com/plugin/?idea_ce&pluginId=7157 插件源码地址：https://github.com/setial/intellij-javadocs 推荐指数： 五星

19. Android strings.xml tools 可以用来管理Android项目中的字符串资源。它提供了排序Android本地文件和添加缺少的字符串的基本操作。虽然这个插件是有限制的，但如果应用程序有大量的字符串资源，那这个插件就非常有用了。 插件下载地址：https://plugins.jetbrains.com/plugin/7498?pr= 插件源码地址：https://github.com/constantine-ivanov/strings-xml-tools 推荐指数：五星

20. Robotium Recorder Robotium Recorder是一个自动化测试框架，用于测试在模拟器和Android设备上原生的和混合的移动应用程序。Robotium Recorder可以让你记录测试案例和用户操作。你也可以查看不同Android活动时的系统功能和用户测试场景。 插件下载地址：https://plugins.jetbrains.com/plugin/7513?pr= 插件官方网址：http://robotium.com/ 推荐指数：四星

21. Android Holo Colors Generator 通过自定义Holo主题颜色生成对应的Drawable和布局文件 插件下载地址：https://plugins.jetbrains.com/plugin/7366?pr= 插件源码地址：https://github.com/jeromevdl/android-holo-colors-idea-plugin 推荐指数：四星

22. lint-cleaner-plugin 删除未使用的资源,包括String字符串,颜色和尺寸。 这是一个Gradle插件，所以如何配置可以去github的源码上看。 插件源码地址：https://github.com/marcoRS/lint-cleaner-plugin 推荐指数：四星

23. codota 该网站搜集了大量的代码，号称超过700W的代码实例。提供了chrome和as插件。 插件下载地址：https://plugins.jetbrains.com/plugin/7638?pr= 插件官方网址：https://www.codota.com/ 推荐指数：五星

24. ECTranslation 一个androidstudio上面的翻译插件（将英文翻译为中文）。 插件下载地址：https://plugins.jetbrains.com/plugin/8469 插件源码地址：https://github.com/Skykai521/ECTranslation 推荐指数：四星

25. TranslationPlugin Android Studio/IntelliJ IDEA 翻译插件,可中英互译。暂时以jar方式安装。 插件源码地址：https://github.com/YiiGuxing/TranslationPlugin 推荐指数：三星

26. Android File Grouping Plugin 该插件可自动将前缀相同的文件归类显示到同一文件目录下，但不会因此而移动文件或创建文件夹。 插件下载地址：https://github.com/dmytrodanylyk/folding-plugin/releases 插件源码地址：https://github.com/dmytrodanylyk/folding-plugin 推荐指数： 四星

27. PermissionsDispatcher 一个针对API 23，可在 `Activity/Fragment` 中快速生成 `Runtime Permissions` 代码的插件。 插件下载地址：https://plugins.jetbrains.com/plugin/8349 插件源码地址：https://github.com/shiraji/permissions-dispatcher-plugin 推荐指数： 四星

28. Android code Generator Android Studio/IntelliJ IDEA的安卓代码生成插件，帮助提高app的开发速度。可以从layout生成Activity类、Fragment类、Adapter类，从menu.xml生成menu代码等。 插件下载地址：https://plugins.jetbrains.com/plugin/7595?pr= 插件源码地址：https://github.com/tmorcinek/android-codegenerator-plugin-intellij 插件教程：（中文版）http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2016/0523/4294.html、（英文版）http://tmorcinek.github.io/android-codegenerator-plugin-intellij/ 推荐指数：五星

29. .ignore 项目中，每次add，commit的时候有可能会把Module生成的一些build文件/本地配置文件/iml文件提交上去。可以通过gitignore解决，如果你不想提交的文件，就可以在创建项目的时候将这个文件中添加即可，将一些通用的东西屏蔽掉。 插件下载地址：https://plugins.jetbrains.com/plugin/7495?pr=androidstudio 插件源码地址：https://github.com/hsz/idea-gitignore 推荐指数：四星

30. checkstyle-idea CheckStyle-IDEA 是一个检查代码风格的插件，比如像命名约定，Javadoc，类设计等方面进行代码规范和风格的检查，你们可以遵从像Google Oracle 的Java 代码指南 ，当然也可以按照自己的规则来设置配置文件，从而有效约束你自己更好地遵循代码编写规范。 插件下载地址：https://plugins.jetbrains.com/plugin/1065?pr=androidstudio 插件源码地址：https://github.com/jshiell/checkstyle-idea 推荐指数：四星

31. Android Methods Count 统计Android依赖库中方法的总个数。 (一个dex只能接受的65K并不是指方法数超过65K而报的错,而是指引用计数超过65K) 插件下载地址：https://plugins.jetbrains.com/plugin/8076?pr=androidstudio 推荐指数：四星

32. Sexy Editor 设置代码性感背景图，还是比较强悍的。 插件下载地址：https://plugins.jetbrains.com/plugin/1833?pr=androidstudio 插件源码地址：https://github.com/igorspasic/idea-sexyeditor 推荐指数：五星

33. AndroidProguardPlugin Android一键生成项目混淆代码插件，现在jetbrains还在审核只能下载进行安装了，不能通过as插件直接搜索安装。因为混淆时很多同学比较头疼的一个事情，所以给5星。 插件下载地址：https://raw.githubusercontent.com/zhonghanwen/AndroidProguardPlugin/master/AndroidProguard.zip 插件源码地址：https://github.com/zhonghanwen/AndroidProguardPlugin 推荐指数：五星

34. Android Studio Prettify 从布局文件一键生成对view的声明（不适用注解，形式为findviewById的方式）,还可以将代码中的字符串写在String.xml文件中。 插件下载地址：https://plugins.jetbrains.com/plugin/7405 插件源码地址：https://github.com/Haehnchen/idea-android-studio-plugin 推荐指数：四星

35. Gradle Dependencies Helper Maven gradle依赖自动补全 插件下载地址：https://plugins.jetbrains.com/plugin/7299 插件源码地址：https://github.com/siosio/GradleDependenciesHelperPlugin 推荐指数：五星

36. Remove ButterKnife ButterKnife这个第三方库每次更新之后，绑定view的注解都会改变，从bind,到inject，再到bindview，一旦升级，就会有巨量的代码需要手动修改,所以这个插件可以快速移除，将注解代码变成findviewbyId的形式。 插件下载地址：https://plugins.jetbrains.com/plugin/8432 插件源码地址：https://github.com/u3shadow/RemoveButterKnife 插件教程地址：https://github.com/u3shadow/RemoveButterKnife/blob/master/README.md 推荐指数：三星

37. Android DPI Calculator Dpi计算插件 插件下载地址：https://plugins.jetbrains.com/plugin/7832 插件源码地址：https://github.com/JerzyPuchalski/Android-DPI-Calculator 推荐指数：五星

38. SingletonTest 快速生成单例模式的插件。单例模式的六种生成方式 LazyUnSafe,LazySafe,Hungry,DoubleCheck,StaticInner,Enum(http://cantellow.iteye.com/blog/838473);插件需要从github中下载jar安装。 插件源码地址：https://github.com/luhaoaimama1/SingletonTest 推荐指数：四星

39. Android Localizationer 将项目中的 string 资源自动翻译为其他语言的 Android Studio/IntelliJ IDEA 插件。插件下载地址：https://plugins.jetbrains.com/plugin/7629 插件源码地址：https://github.com/westlinkin/AndroidLocalizationer 推荐指数：四星

40. Material Theme UI 添加Material主题到你的AS 插件下载地址：https://plugins.jetbrains.com/plugin/8006?pr= 插件源码地址：https://github.com/ChrisRM/material-theme-jetbrains 推荐指数：五星

41. gradle-retrolambda 在java6,java7中也能使用Lambda表达式。这个使用方式跟其他插件不同，使用方式请看github的说明。 插件源码地址：https://github.com/evant/gradle-retrolambda 推荐指数：五星

42. eventbus-intellij-plugin eventbus导航插件，方便快速查找，但是有2个pr没处理，已经有1年没维护更新了。 插件源码地址：https://github.com/kgmyshin/eventbus-intellij-plugin 推荐指数：三星

43. otto-intellij-plugin otto导航插件，也是很久没维护了，不过可以用。 插件源码地址：https://github.com/square/otto-intellij-plugin 推荐指数：三星

44. dagger-intellij-plugin dagger可视化辅助工具 插件源码地址：https://github.com/square/dagger-intellij-plugin 推荐指数：四星

45. Android Styler 根据xml文件生成style文件的插件 插件下载地址：https://plugins.jetbrains.com/plugin/7972 插件源码地址：https://github.com/alexzaitsev/android-styler 推荐指数：四星
46. ideavim 能让开发使用vi，大大提高开发效率。 插件下载地址：https://plugins.jetbrains.com/plugin/164?pr= 插件源码地址：https://github.com/JetBrains/ideavim 推荐指数：五星
47. Android Material Design Icon Generator 可以在Android项目中设置 material design样式图标(material design icons) 插件下载地址：https://plugins.jetbrains.com/plugin/7647?pr= 插件源码地址：https://github.com/konifar/android-material-design-icon-generator-plugin 推荐指数： 五星
48. Gradle Killer Gradle 卡死可以用这个插件。 插件下载地址：https://plugins.jetbrains.com/plugin/7794?pr= 插件源码地址：https://github.com/KanbanApps/GradleKillerIdeaPlugin 推荐指数：五星
49. android-toolbox-plugin 能根据xml文件生成对应的ViewHolder类，可以是findviewbyId形式或者是ButterKnife刀形式，如果你使用ButterKnife。 插件下载地址：https://plugins.jetbrains.com/plugin/7200 插件源码地址：https://github.com/idamobile/android-toolbox-plugin 推荐指数：三星
50. java2smali 能快速将当前java文件编译成smali文件，方便学习smali语法的童鞋来对比源码学习。 插件下载地址：https://plugins.jetbrains.com/plugin/7385 插件源码地址：https://github.com/ollide/intellij-java2smali 推荐指数：三星
51. PermissionsDispatcher plugin 自动生成android6.0权限代码 插件下载地址：https://plugins.jetbrains.com/plugin/8349 插件源码地址：https://github.com/shiraji/permissions-dispatcher-plugin 推荐指数：五星
52. WakaTime 记录你在IDE上工作的时间 插件下载地址：https://plugins.jetbrains.com/plugin/7425 插件源码地址：https://github.com/wakatime/jetbrains-wakatime 推荐指数：五星
53. Exynap Exynap是一个帮助你快速查找和完成代码插入的AndroidStudio插件。你只需输入一个命令按回车就能调出一段代码，而这个命令也不需要你去记，只需输入大致意思就可以了，Exynap提供了上千种智能代码解决方案。 插件下载地址：https://plugins.jetbrains.com/plugin/8600?pr=idea 插件主页地址：http://exynap.com/ 插件教程地址：http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2016/0908/6606.html 推荐指数：五星

插件开发

一、概述

相信大家在使用Android Studio的时候，或多或少的会使用一些插件，适当的配合插件可以帮助我们提升一定的开发效率，更加快乐。例如：

https:\/\/github.com\/zzz40500\/GsonFormat 可以帮助我们从Gson转化为实体类https:\/\/github.com\/avast\/android-butterknife-zelezny 可以帮助我们更加方便的使用butterknifehttps:\/\/github.com\/Skykai521\/ECTranslation 可以帮助在IDE里面完成英文->中文的翻译

有句话叫做授人以鱼不如授人以渔，不能一直跟随着别人的脚步去使用插件了，有必要去学习编写插件，当自己有好的创意的时候，就可以自己实现了。So，本文的内容是：

自己编写一个Android Studio插件

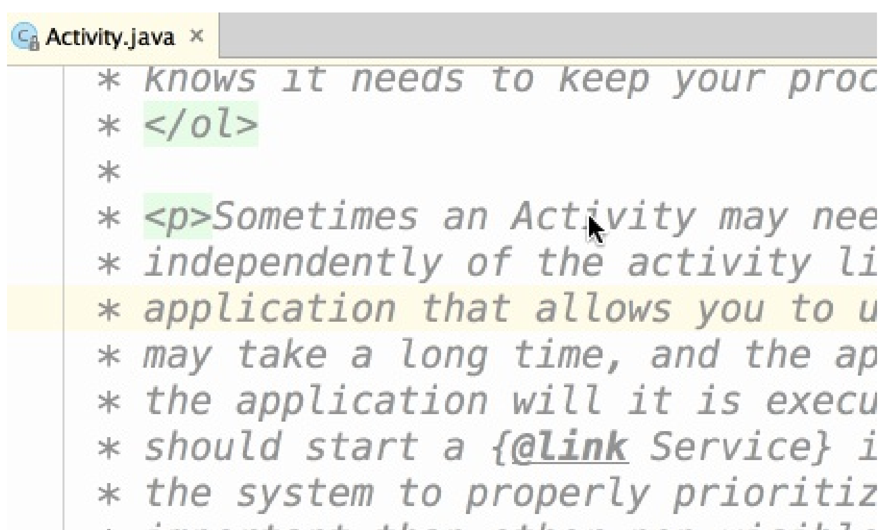
ok，其实编写插件并不难，官方也有详细的文档，所以你也可以选择直接阅读下文学习：

http:\/\/www.jetbrains.org\/intellij\/sdk\/docs\/basics\/getting_started.html

为了文章有一定的流畅性，决定以ECTranslation作为编写Android Studio插件的例子。

我为什么选这个呢？因为创意好，实用并且代码简单。

贴一个今天这个插件的最终效果图：



注：效果与ECTranslation基本一致，本文仅用作学习，不造轮子，如果需要使用，直接使用ECTranslation即可。

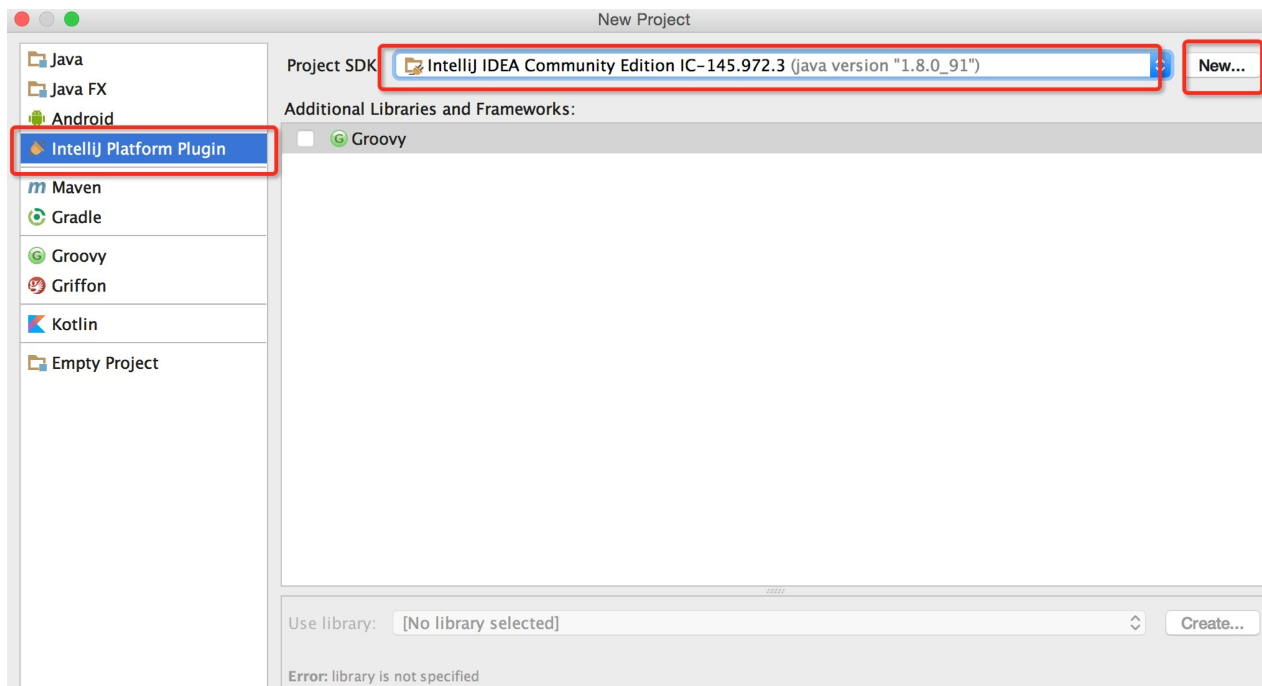
二、准备工作

首先需要安装IntelliJ IDEA

下载网址：https:\/\/www.jetbrains.com\/idea\/

下载好就可以了~~

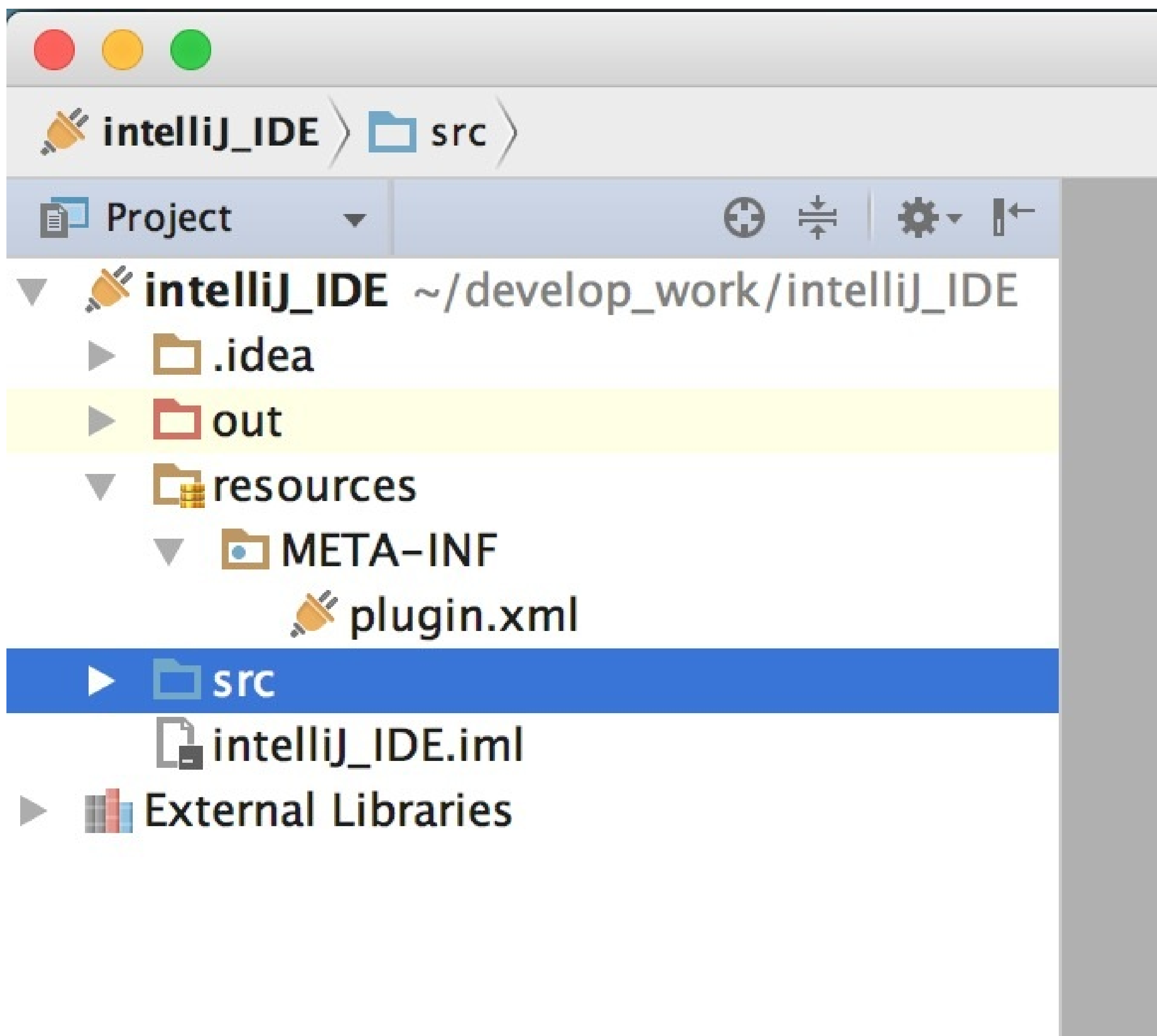
然后安装，运行，点击create New Project:



按照上图进行选择，如果没有SDK，则点击New新建一个即可。

然后点击Next,输入项目名称选择位置，就可以点击finish了。

项目的结构如下：



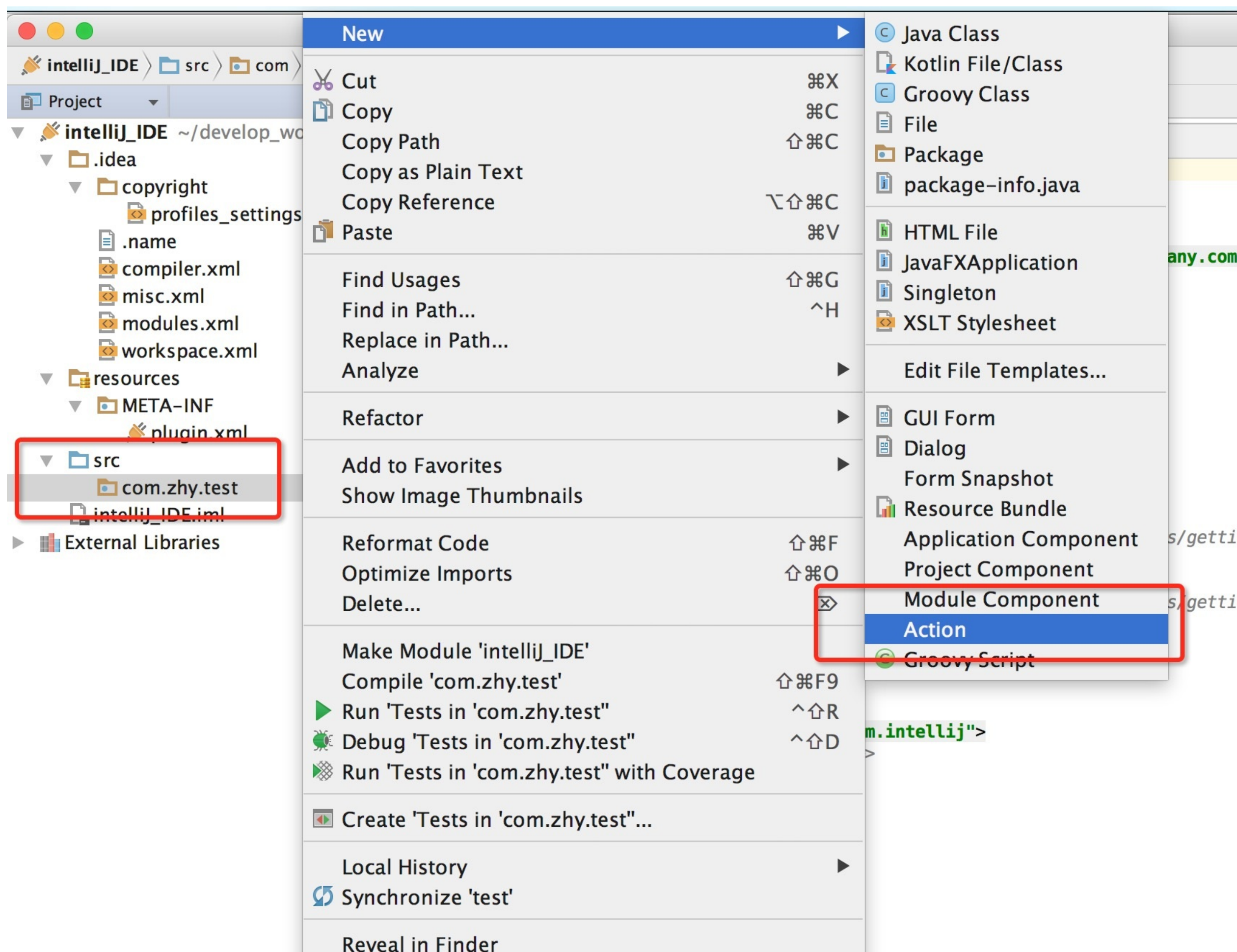
src目录下主要用于存放我们编写的代码。

这样准备工作就结束了~~

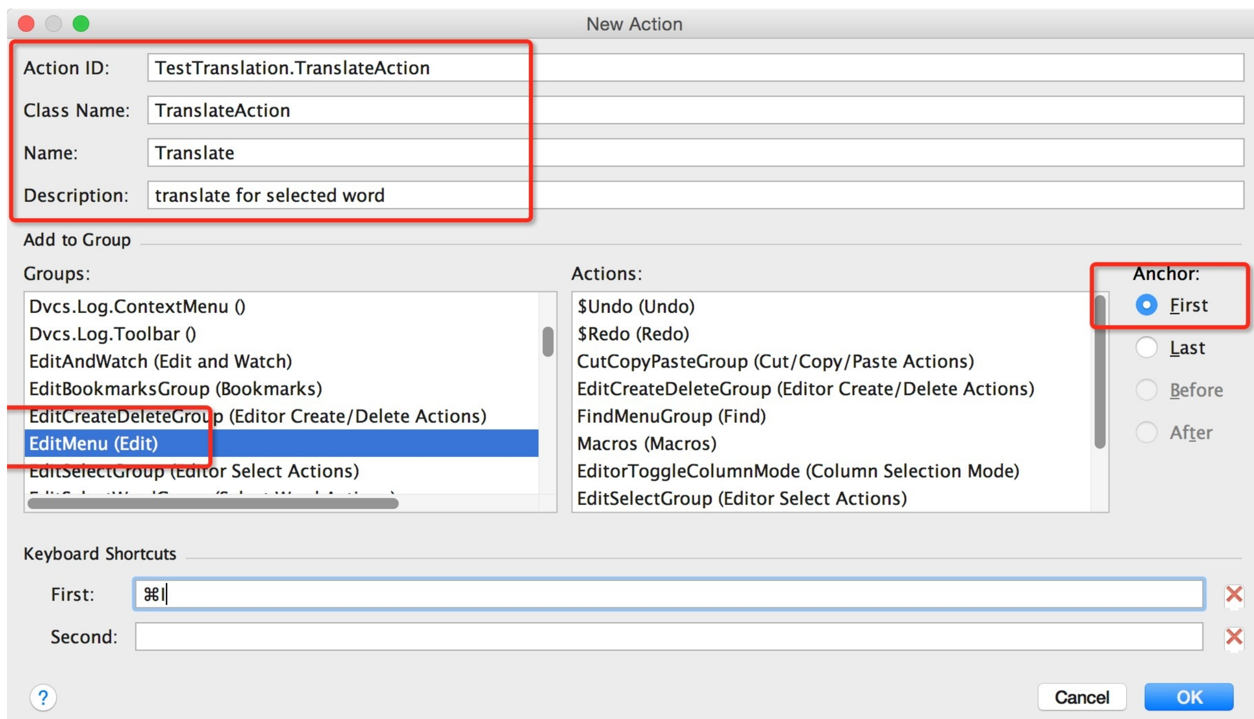
三、编码

(1) 关键知识

编码实际上核心的一个类叫做AnAction，可以直接选择NEW->Action，如下图：



然后填写一些相关信息：

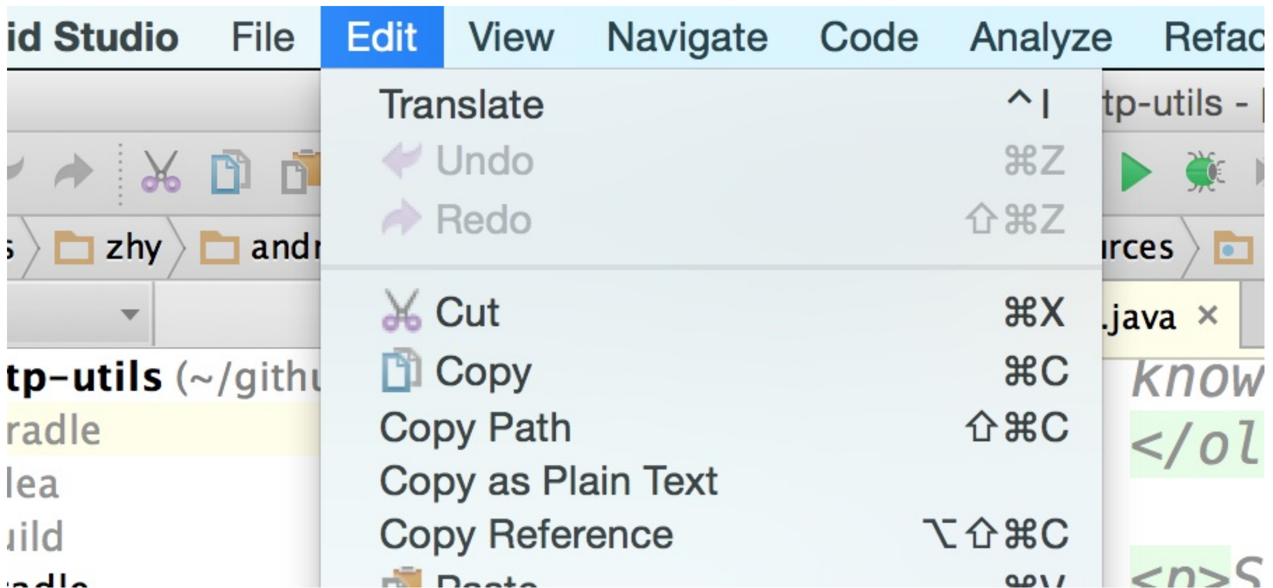


需要填写的属性如下：

ActionID:代表该Action的唯一的ID，一般的格式为：pluginName.ID ClassName:类名 Name:就是最终插件在菜单上的名称 Description:对这个Action的描述信息

然后往下，选择这个Action即将存在的位置：

我们选择的是EditMenu，右侧选择为first，即EditMenu下的第一个，效果如图：



再往下就是制定快捷键了~~

都填写完成就可以点击OK了。

点击ok之后，可以看到为我们生成了下类：

```
public class TranslateAction extends AnAction {

@Override

public void actionPerformed(AnActionEvent e) {

// TODO: insert action logic here

}

}
```

此外我们刚才填写的信息，也在plugin.xml中完成了注册，大家可以进去看一眼，actions的标签中，

当我们点击菜单的时候，就回触发 `actionPerformed()` 方法。

那么这么看，我们在这个方法中只要完成三件事：

获得当前选中的单词 调用相关API得到单词的意思 通过一个类似于PopupWindow来显示

当然，为了尽快的测试，你可以先在里面弹一个对话框，例如如下：

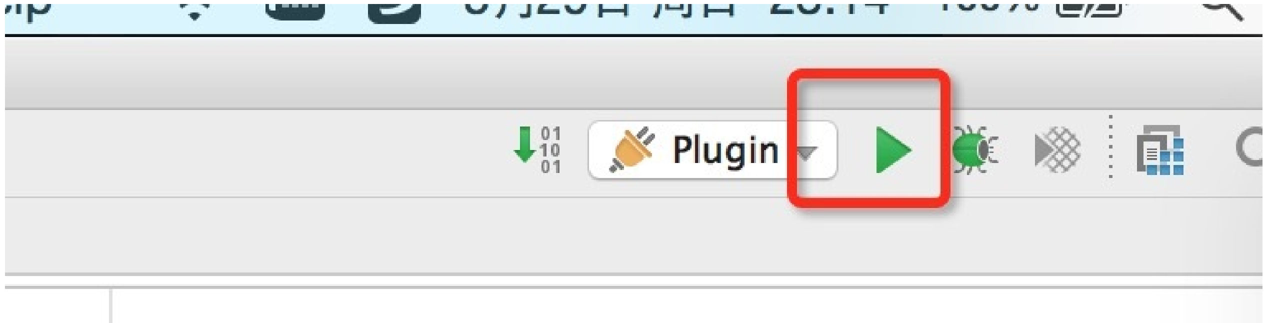
```
public void actionPerformed(AnActionEvent event) {

Messages.showMessageDialog("Hello World !", "Information", Messages.getInformationIcon
());

}
```

预期效果是点击Tranlate菜单，或者按快捷键会弹出一个提示对话框。

那么点击Run:



然后它会默认启动一个新的IntelliJ IDEA的界面，你可以随便新建一个项目，进入以后，你会发现Edit下多了一个Translate菜单，点击即可弹出我们设定的对话框：



ok，测试通过就放心了~

> 获得当前选中的单词 调用相关API得到单词的意思 通过一个类似于PopupWindow来显示

剩下的就是功能性的API了~

(2) 获得当前选中的单词

```
@Override

public void actionPerformed(AnActionEvent e) {

// TODO: insert action logic here

final Editor mEditor = e.getData(PlatformDataKeys.EDITOR);

if (null == mEditor) {

return;

}

SelectionModel model = mEditor.getSelectionModel();

final String selectedText = model.getSelectedText();

if (TextUtils.isEmpty(selectedText)) {

return;

}

}
```

是不是觉得API很陌生，恩，我也觉得很陌生，关于API这里介绍其实没什么意义，本文主要目的是让大家对自定义插件有个类helloworld的认识，至于插件里面的代码涉及到的API等到大家需要编写插件的时候，再详细学习就好了，现在就不要浪费精力记忆这些东西了。

上面的代码就是获得选中的文本，通过一个Editor，然后拿到SelectionModel，再拿到selectedText，从字面上还是蛮好理解的。

拿到选中的文本之后，应该就是去查询该单词的意思了，查询呢，ECTranslation用的是youdao的Open SDK，其实也很简单，就是拼接一个url，然后等着解析返回数据就好了。

（3）调用相关API得到单词的意思

有道API的地址：

http:\/\/fanyi.youdao.com\/openapi?path=data-mode

大家如果想要做单词翻译，可以看下，非常简单。

涉及到的代码：

```
String baseUrl = "http://fanyi.youdao.com/openapi.do?keyfrom=Skykai521&key=977124034&t
ype=data&doctype=json&version=1.1&q=";

HttpUtils.doGetAsyn(baseUrl + selectedText, new HttpUtils.CallBack() {

public void onRequestComplete(String result) {

Translation translation = gson.fromJson(result, Translation.class);

showPopupBalloon(mEditor, translation.toString());

}

});
```

HttpUtils就不贴了，就是直接开了个线程，通过HttpUrlConnection去访问网络，大家的项目中或者通过搜索引擎，代码一搜一堆。

baseUrl就是有道的url，加上我们选中的单词就是完整的url了，然后通过http访问，callback回调出返回的字符串，这里返回的是json类型的字符串。

baseUri是：

http:\/\/fanyi.youdao.com\/openapi.do?keyfrom=Skykai521&key=977124034&type=data&doctype=json&version=1.1&q=name

我们根据返回的json字符串生成了一个类Translation；

然后通过Gson转化为Translation对象。

ps:拿着上面的baseUrl后面跟一个任何单词，直接访问浏览器就能看到返回的json数据了，这里大家天天写接口，类似的步骤比我肯定还熟悉。

好了，有了返回的数据以后，直接通过一个类似popupWindow展现即可。

（4）通过一个类似于PopupWindow来显示

涉及到的代码：

```
private void showPopupBalloon(final Editor editor, final String result) {

ApplicationManager.getApplication().invokeLater(new Runnable() {

public void run() {

JBPopupFactory factory = JBPopupFactory.getInstance();

factory.createHtmlTextBalloonBuilder(result, null, new JBColor(new Color(186, 238, 186
), new Color(73, 117, 73)), null)

.setFadeoutTime(5000)

.createBalloon()

.show(factory.guessBestPopupLocation(editor), Balloon.Position.below);

}

});

}
```

这个API，恩，我copy的源码，依然是不求记住，知道这有个类似的功能即可。

简单看一下，是通过创建一个JBPopupFactory，然后通过它创建一个HtmlTextBalloonBuilder，通过这个builder去设置各种参数，最后show。

ok，对于一个入门的例子，不要太强求对插件中这些API的掌握，还是那句话，等需要写了再去查，需要什么功能，哪怕到对应的插件中去copy源码都可以，当然也有文档：

http:\/\/www.jetbrains.org\/intellij\/sdk\/docs\/tutorials.html

有兴趣的可以整理各种类型的插件，比如弹出popupWindow，生成代码，生成文件类别的，然后对相关的API进行收集与整理。

这样代码写完了，先测试一下，点击RUN，然后看效果~

我们这里肯定是测试没问题的，效果图就是开始的那个gif.

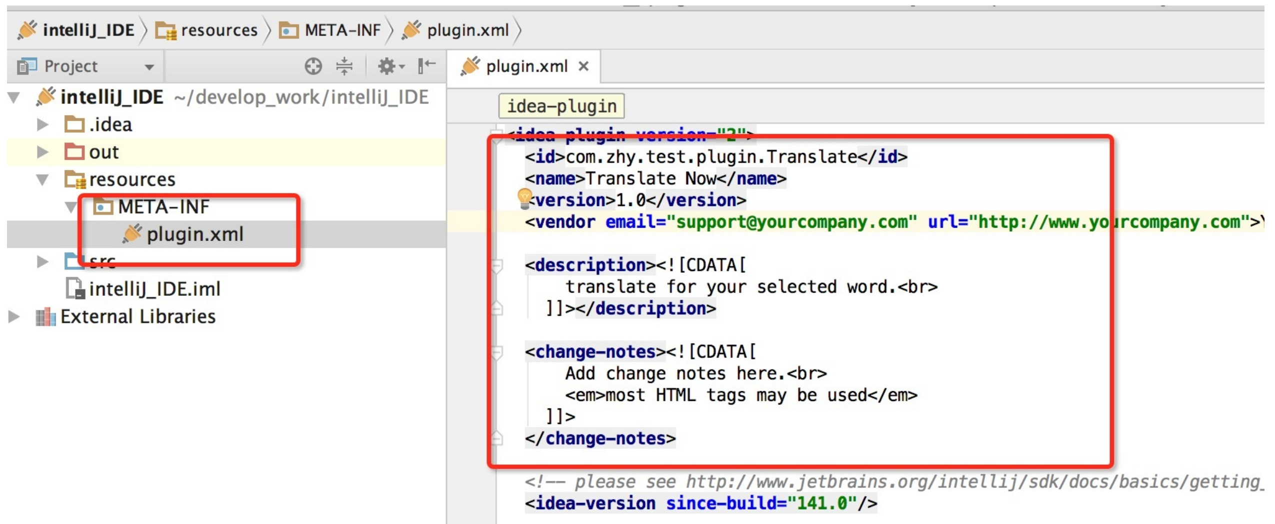
如果没有问题，就可以去部署和发布我们的插件给别人去使用了。

这两部也非常简单。

四、部署插件

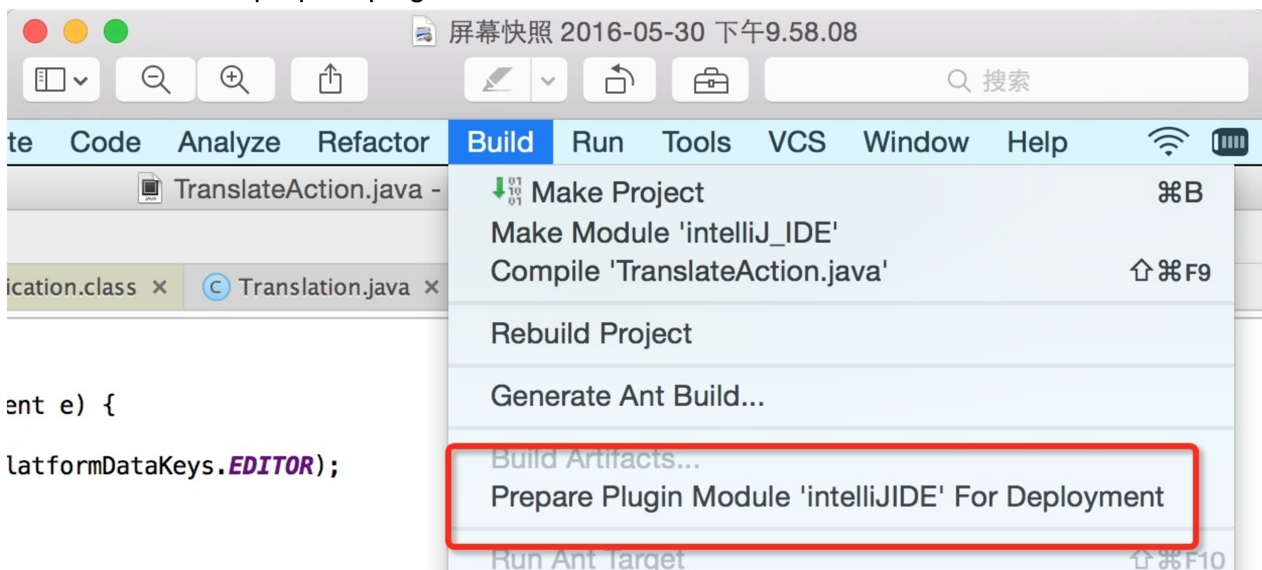
（1）填写插件相关信息

打开项目文件的plugin.xml，如下图：

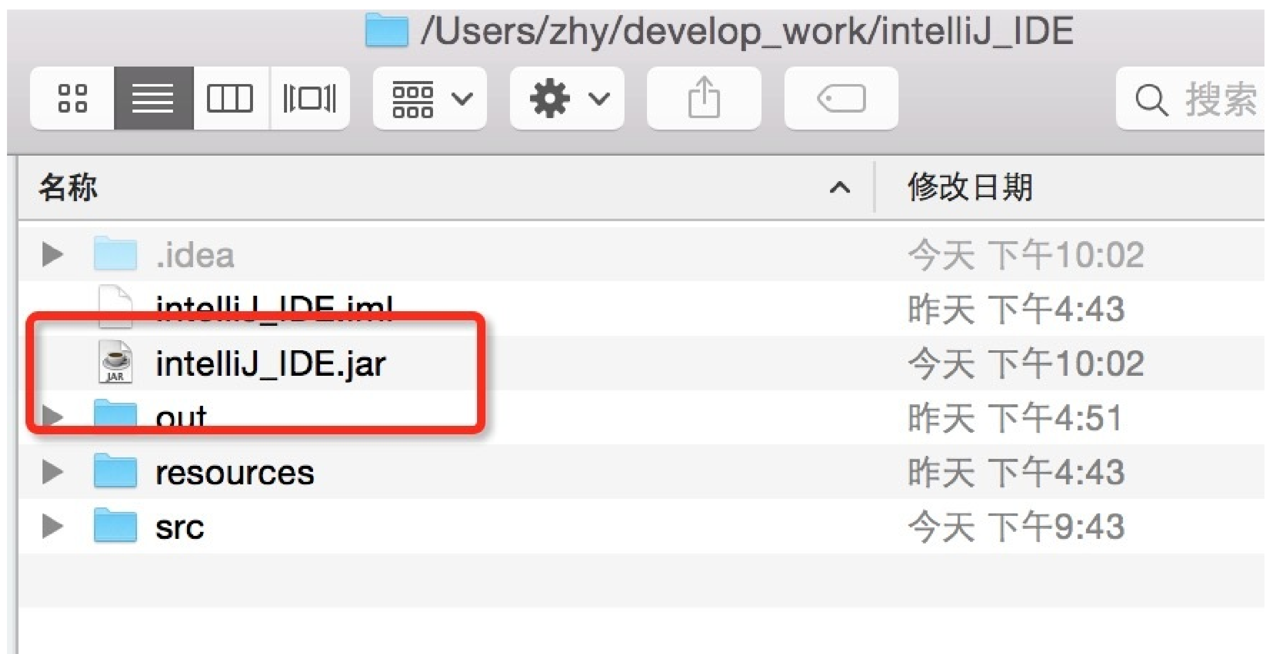


在里面填写id,name,version等。。。记得随便填一下~

然后，点击build->prepare plugin...，如下图：



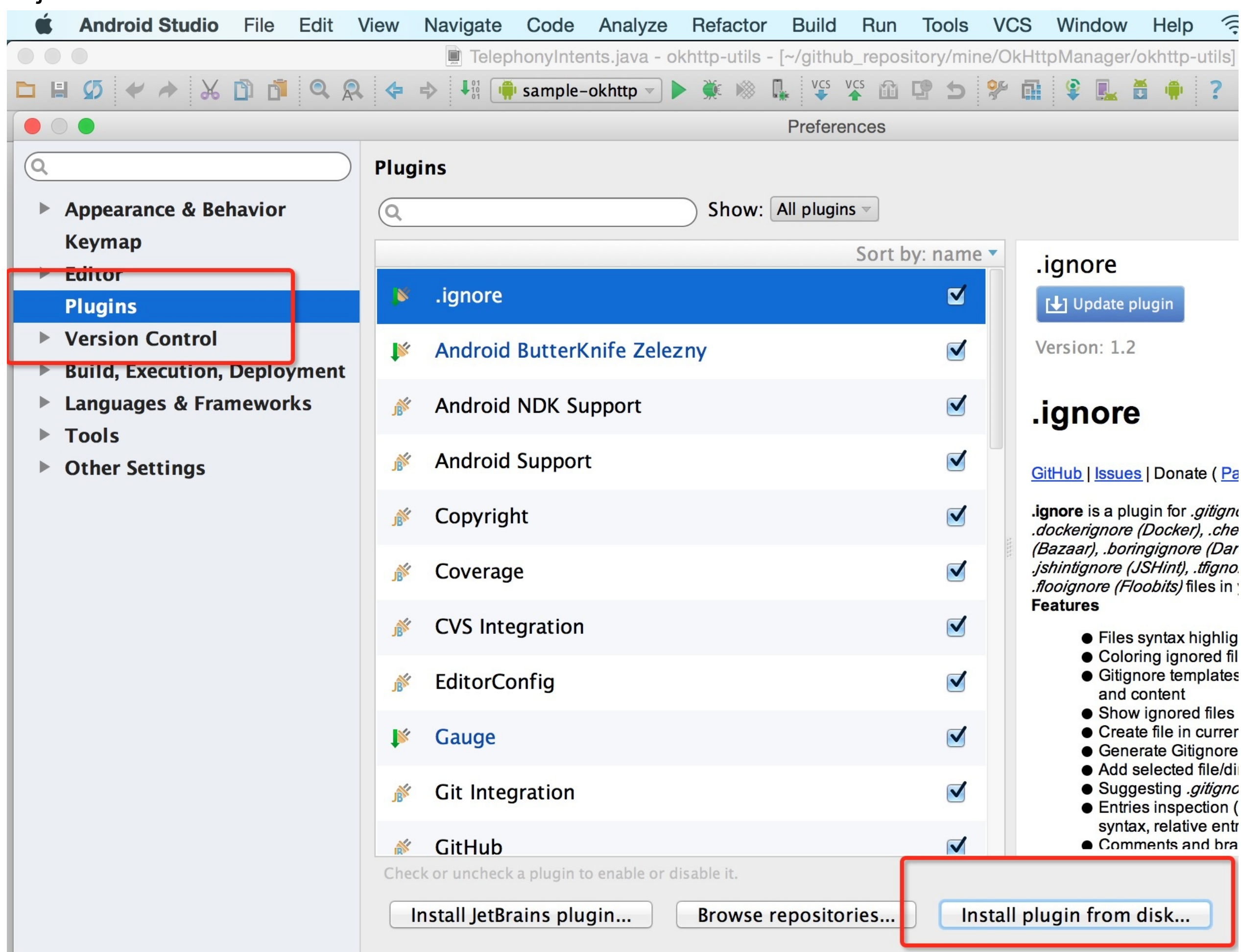
会在项目的根目录生成一个jar，如图：



这个jar就可以用于安装了。

（2）安装插件

打开Andorid Studio，选择Preferences -> Plugins -> Install plugin from disk，选择我们生成的jar即可，如图：



点击安装，然后重启即可。

好了，重启完成就可以在EDIT下看到Translate菜单了，选中单词，点击菜单或者快捷键都能实现翻译了。

如果你有兴趣，赶紧编写一个插件自己玩吧。

当然，还可以把我们的插件发布到仓库，支持在plugin中搜索安装，参考：

http:\/\/www.jetbrains.org\/intellij\/sdk\/docs\/basics\/getting_started\/publishing_plugin.html

就是注册账号，提交jar，填写信息，等着审核就可以了。

五、总结

终于到了总结的环节，这么长的文章其实编写插件总结起来就几句话。

下载Intellij IDEA，新建一个Intellij IDEA plugin的项目 然后在里面new Action以及编写API 点击prepare plugin生成jar，这个jar就可以用来安装了。

恩，就是这么简单，实践起来会比较麻烦一点，等成功以后，回过头来总结，发现步骤其实就那么几个步骤~~对于实际的Action相关的API，等你在编写相关插件的时候，参考别的类似插件，查看官方文档都可以。

第十七章 帮助

查找操作功能

效率指南

附录 快捷键

Ctrl

快捷键	介绍
Ctrl + F	在当前文件进行文本查找 （必备）
Ctrl + R	在当前文件进行文本替换 （必备）
Ctrl + Z	撤销 （必备）
Ctrl + Y	删除光标所在行 或 删除选中的行 （必备）
Ctrl + X	剪切光标所在行 或 剪切选择内容
Ctrl + C	复制光标所在行 或 复制选择内容
Ctrl + D	复制光标所在行 或 复制选择内容，并把复制内容插入光标位置下面 （必备）
Ctrl + W	递进式选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展选中范围 （必备）
Ctrl + E	显示最近打开的文件记录列表
Ctrl + N	根据输入的 类名 查找类文件
Ctrl + G	在当前文件跳转到指定行处
Ctrl + J	插入自定义动态代码模板
Ctrl + P	方法参数提示显示
Ctrl + Q	光标所在的变量 / 类名 / 方法名等上面（也可以在提示补充的时候按），显示文档内容
Ctrl + U	前往当前光标所在的方法的父类的方法 / 接口定义
Ctrl + B	进入光标所在的方法/变量的接口或是定义处，等效于 Ctrl + 左键单击
Ctrl + K	版本控制提交项目，需要此项目有加入到版本控制才可用
Ctrl + T	版本控制更新项目，需要此项目有加入到版本控制才可用
Ctrl + H	显示当前类的层次结构
Ctrl + O	选择可重写的方法
Ctrl + I	选择可继承的方法
Ctrl + +	展开代码
Ctrl + -	折叠代码

Ctrl + /	注释光标所在行代码，会根据当前不同文件类型使用不同的注释符号 （必备）
Ctrl + [	移动光标到当前所在代码的花括号开始位置
Ctrl +]	移动光标到当前所在代码的花括号结束位置
Ctrl + F1	在光标所在的错误代码处显示错误信息
Ctrl + F3	调转到所选中的词的下一个引用位置
Ctrl + F4	关闭当前编辑文件
Ctrl + F8	在 Debug 模式下，设置光标当前行为断点，如果当前已经是断点则去掉断点
Ctrl + F9	执行 Make Project 操作
Ctrl + F11	选中文件 / 文件夹，使用助记符设定 / 取消书签
Ctrl + F12	弹出当前文件结构层，可以在弹出的层上直接输入，进行筛选
Ctrl + Tab	编辑窗口切换，如果在切换的过程又加按上delete，则是关闭对应选中的窗口
Ctrl + Enter	智能分隔行
Ctrl + End	跳到文件尾
Ctrl + Home	跳到文件头
Ctrl + Space	基础代码补全，默认在 Windows 系统上被输入法占用，需要进行修改，建议修改为 Ctrl + 逗号 （必备）
Ctrl + Delete	删除光标后面的单词或是中文句
Ctrl + BackSpace	删除光标前面的单词或是中文句
Ctrl + 1,2,3...9	定位到对应数值的书签位置
Ctrl + 左键单击	在打开的文件标题上，弹出该文件路径
Ctrl + 光标定位	按 Ctrl 不要松开，会显示光标所在的类信息摘要
Ctrl + 左方向键	光标跳转到当前单词 / 中文句的左侧开头位置
Ctrl + 右方向键	光标跳转到当前单词 / 中文句的右侧开头位置
Ctrl + 前方向键	等效于鼠标滚轮向前效果

Ctrl + 后方向键	等效于鼠标滚轮向后效果

Alt

快捷键	介绍
Alt + `	显示版本控制常用操作菜单弹出层
Alt + Q	弹出一个提示，显示当前类的声明 / 上下文信息
Alt + F1	显示当前文件选择目标弹出层，弹出层中有很多目标可以进行选择
Alt + F2	对于前面页面，显示各类浏览器打开目标选择弹出层
Alt + F3	选中文本，逐个往下查找相同文本，并高亮显示
Alt + F7	查找光标所在的方法 / 变量 / 类被调用的地方
Alt + F8	在 Debug 的状态下，选中对象，弹出可输入计算表达式调试框，查看该输入内容的调试结果
Alt + Home	定位 / 显示到当前文件的 Navigation Bar
Alt + Enter	IntelliJ IDEA 根据光标所在问题，提供快速修复选择，光标放在的位置不同提示的结果也不同（必备）
Alt + Insert	代码自动生成，如生成对象的 set / get 方法，构造函数，toString() 等
Alt + 左方向键	按左方向切换当前已打开的文件视图
Alt + 右方向键	按右方向切换当前已打开的文件视图
Alt + 前方向键	当前光标跳转到当前文件的前一个方法名位置
Alt + 后方向键	当前光标跳转到当前文件的后一个方法名位置
Alt + 1,2,3...9	显示对应数值的选项卡，其中 1 是 Project 用得最多

Shift

快捷键	介绍
Shift + F1	如果有外部文档可以连接外部文档
Shift + F2	跳转到上一个高亮错误 或 警告位置
Shift + F3	在查找模式下，查找匹配上一个
Shift + F4	对当前打开的文件，使用新Windows窗口打开，旧窗口保留
Shift + F6	对文件 / 文件夹 重命名
Shift + F7	在 Debug 模式下，智能步入。断点所在行上有多个方法调用，会弹出进入哪个方法
Shift + F8	在 Debug 模式下，跳出，表现出来的效果跟 F9 一样
Shift + F9	等效于点击工具栏的 Debug 按钮
Shift + F10	等效于点击工具栏的 Run 按钮
Shift + F11	弹出书签显示层
Shift + Tab	取消缩进
Shift + ESC	隐藏当前 或 最后一个激活的工具窗口
Shift + End	选中光标到当前行尾位置
Shift + Home	选中光标到当前行头位置
Shift + Enter	开始新一行。光标所在行下空出一行，光标定位到新行位置
Shift + 左键单击	在打开的文件名上按此快捷键，可以关闭当前打开文件
Shift + 滚轮前后滚动	当前文件的横向滚动轴滚动

Ctrl + Alt

快捷键	介绍
Ctrl + Alt + L	格式化代码，可以对当前文件和整个包目录使用 （必备）
Ctrl + Alt + O	优化导入的类，可以对当前文件和整个包目录使用 （必备）
Ctrl + Alt + I	光标所在行 或 选中部分进行自动代码缩进，有点类似格式化
Ctrl + Alt + T	对选中的代码弹出环绕选项弹出层
Ctrl + Alt + J	弹出模板选择窗口，将选定的代码加入动态模板中
Ctrl + Alt + H	调用层次
Ctrl + Alt + B	在某个调用的方法名上使用会跳到具体的实现处，可以跳过接口
Ctrl + Alt + V	快速引进变量
Ctrl + Alt + Y	同步、刷新
Ctrl + Alt + S	打开 IntelliJ IDEA 系统设置
Ctrl + Alt + F7	显示使用的地方。寻找被该类或是变量被调用的地方，用弹出框的方式找出来
Ctrl + Alt + F11	切换全屏模式
Ctrl + Alt + Enter	光标所在行上空出一行，光标定位到新行
Ctrl + Alt + Home	弹出跟当前文件有关联的文件弹出层
Ctrl + Alt + Space	类名自动完成
Ctrl + Alt + 左方向键	退回到上一个操作的地方 （必备）
Ctrl + Alt + 右方向键	前进到上一个操作的地方 （必备）
Ctrl + Alt + 前方向键	在查找模式下，跳到上个查找的文件
Ctrl + Alt + 后方向键	在查找模式下，跳到下个查找的文件

Ctrl + Shift

快捷键	介绍
Ctrl + Shift + F	根据输入内容查找整个项目 或 指定目录内文件 （必备）
Ctrl + Shift + R	根据输入内容替换对应内容，范围为整个项目 或 指定目录内文件 （必备）
Ctrl + Shift +	自动将下一行合并到当前行末尾 （必备）

J	自动将下一行合并到当前行末尾 （必备）
Ctrl + Shift + Z	取消撤销 （必备）
Ctrl + Shift + W	递进式取消选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展取消选中范围 （必备）
Ctrl + Shift + N	通过文件名定位 / 打开文件 / 目录，打开目录需要在输入的内容后面多加一个正斜杠（必备）
Ctrl + Shift + U	对选中的代码进行大 / 小写轮流转换 （必备）
Ctrl + Shift + T	对当前类生成单元测试类，如果已经存在的单元测试类则可以进行选择
Ctrl + Shift + C	复制当前文件磁盘路径到剪贴板
Ctrl + Shift + V	弹出缓存的最近拷贝的内容管理器弹出层
Ctrl + Shift + E	显示最近修改的文件列表的弹出层
Ctrl + Shift + H	显示方法层次结构
Ctrl + Shift + B	跳转到类型声明处
Ctrl + Shift + I	快速查看光标所在的方法 或 类的定义
Ctrl + Shift + A	查找动作 / 设置
Ctrl + Shift + /	代码块注释 （必备）
Ctrl + Shift + [	选中从光标所在位置到它的顶部中括号位置
Ctrl + Shift +]	选中从光标所在位置到它的底部中括号位置
Ctrl + Shift + +	展开所有代码
Ctrl + Shift + -	折叠所有代码
Ctrl + Shift + F7	高亮显示所有该选中文本，按Esc高亮消失
Ctrl + Shift + F8	在 Debug 模式下，指定断点进入条件

Ctrl + Shift + F9	编译选中的文件 / 包 / Module
Ctrl + Shift + F12	编辑器最大化
Ctrl + Shift + Space	智能代码提示
Ctrl + Shift + Enter	自动结束代码，行末自动添加分号 （必备）
Ctrl + Shift + Backspace	退回到上次修改的地方
Ctrl + Shift + 1,2,3...9	快速添加指定数值的书签
Ctrl + Shift + 左键单击	把光标放在某个类变量上，按此快捷键可以直接定位到该类中 （必备）
Ctrl + Shift + 左方向键	在代码文件上，光标跳转到当前单词 / 中文句的左侧开头位置，同时选中该单词 / 中文句
Ctrl + Shift + 右方向键	在代码文件上，光标跳转到当前单词 / 中文句的右侧开头位置，同时选中该单词 / 中文句
Ctrl + Shift + 左方向键	在光标焦点是在工具选项卡上，缩小选项卡区域
Ctrl + Shift + 右方向键	在光标焦点是在工具选项卡上，扩大选项卡区域
Ctrl + Shift + 前方向键	光标放在方法名上，将方法移动到上一个方法前面，调整方法排序
Ctrl + Shift + 后方向键	光标放在方法名上，将方法移动到下一个方法前面，调整方法排序

Alt + Shift

快捷键	介绍
Alt + Shift + N	选择 / 添加 task
Alt + Shift + F	显示添加到收藏夹弹出层 / 添加到收藏夹
Alt + Shift + C	查看最近操作项目的变化情况列表
Alt + Shift + I	查看项目当前文件
Alt + Shift + F7	在 Debug 模式下，下一步，进入当前方法体内，如果方法体还有方法，则会进入该内嵌的方法中，依此循环进入
Alt + Shift + F9	弹出 Debug 的可选择菜单
Alt + Shift + F10	弹出 Run 的可选择菜单
Alt + Shift + 左键双击	选择被双击的单词 / 中文句，按住不放，可以同时选择其他单词 / 中文句
Alt + Shift + 前方向键	移动光标所在行向上移动
Alt + Shift + 后方向键	移动光标所在行向下移动

Ctrl + Shift + Alt

快捷键	介绍
Ctrl + Shift + Alt + V	无格式黏贴
Ctrl + Shift + Alt + N	前往指定的变量 / 方法
Ctrl + Shift + Alt + S	打开当前项目设置
Ctrl + Shift + Alt + C	复制参考信息

其他

快捷键	介绍
F2	跳转到下一个高亮错误 或 警告位置 （必备）
F3	在查找模式下，定位到下一个匹配处
F4	编辑源
F7	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果该方法体还有方法，则不会进入该内嵌的方法中
F8	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则不进入当前方法体内
F9	在 Debug 模式下，恢复程序运行，但是如果该断点下面代码还有断点则停在下一个断点上
F11	添加书签
F12	回到前一个工具窗口
Tab	缩进
ESC	从工具窗口进入代码文件窗口
连按两次 Shift	弹出 Search Everywhere 弹出层